PrimeTime® Suite
Variables and Attributes

Version O-2018.06, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Trademarks

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

Free and Open-Source Licensing Notices

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

Third-Party Links

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

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

Copyright Notice for the Command-Line Editing Feature

Copyright Notice for the Line-Editing Library

This software is not subject to any license of the American Telephone and Telegraph Company or of the Regents of the University of California.

Permission is granted to anyone to use this software for any purpose on any computer system, and to alter it and redistribute it freely, subject to the following restrictions:

1. The authors are not responsible for the consequences of use of this software, no matter how awful, even if they arise from flaws in it.
2. The origin of this software must not be misrepresented, either by explicit claim or by omission. Since few users ever read sources, credits must appear in the documentation.
3. Altered versions must be plainly marked as such, and must not be misrepresented as being the original software. Since few users ever read sources, credits must appear in the documentation.
4. This notice may not be removed or altered.

Copyright Notice for the jemalloc Memory Allocator

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice(s), this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice(s), this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDER(S) "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT HOLDER(S) BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright Notice for the CDPL Common Module

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
- Neither the name of Redis nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Contents

arch

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

SEE ALSO

sh_arch(3)

attributes

This man page provides general information about attributes.

DESCRIPTION

An attribute carries information about an object. For example, the **number_of_pins** attribute attached to a cell object indicates the number of pins in the cell.

You can use the following types of attributes:

- Application attributes - Predefined by the tool.
- User-defined attributes - Defined, set, and removed by the **define_user_attribute**, **set_user_attribute**, and **remove_user_attribute** commands, respectively.

To see the value of an attribute, use the **get_attribute** or **report_attribute** command. To list attributes, use the **get_defined_attributes**, **help_attributes**, or **list_attributes** command.

For details about the attributes of an object class, see the man page with "_attributes" appended to the object class name. For example, to see information about design attributes, use this command:

```
pt_shell> man design_attributes
```

SEE ALSO

```
define_user_attribute(2)
get_attribute(2)
get_defined_attributes(2)
help_attributes(2)
list_attributes(2)
remove_user_attribute(2)
set_user_attribute(2)
report_attribute(2)
category_node_attributes(3)
category_tree_attributes(3)
cell_attributes(3)
clock_attributes(3)
correlation_attributes(3)
design_attributes(3)
gui_object_attributes(3)
lib_attributes(3)
lib_cell_attributes(3)
```

lib_pg_pin_info_attributes(3)
lib_pin_attributes(3)
lib_timing_arc_attributes(3)
mode_attributes(3)
net_attributes(3)
path_group_attributes(3)
pg_pin_info_attributes(3)
pin_attributes(3)
port_attributes(3)
scenario_attributes(3)
timing_arc_attributes(3)
timing_path_attributes(3)
timing_point_attributes(3)
upf_power_domain_attributes(3)
upf_power_switch_attributes(3)
upf_supply_net_attributes(3)
upf_supply_port_attributes(3)
upf_supply_set_attributes(3)
variation_attributes(3)

auto_fixable_violations

This man page describes **auto_fixable** hierarchical boundary check violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

The tool can automatically resolve **auto_fixable** boundary violations during subsequent iterations of block- and top-level HyperScale analysis. **auto_fixable** violations are for timing closure or signoff with the HyperScale flow. You should fix these violations after resolving the more critical **non_auto_fixable** violations.

The following violations are **auto_fixable**:

```
boundary_logic_value
clock_latency
clock_skew_with_uncertainty
data_arrival
input_slews
```

WHAT NEXT

To resolve an **auto_fixable** violation, do one of the following actions:

- Rerun the analysis of the block where violations occur with the latest top-level context, and reoptimize the block if necessary.
- Optimize the top-level design to fit the block design scope.

EXAMPLES

The following example shows a verbose report that shows only **auto_fixable** HyperScale boundary violations.

```
pt_shell> report_constraint -boundary_check_include {auto_fixable} \
          -all_violators -verbose
```

```
****************************************
Report : constraint
        -all_violators
        -verbose
Design : top
 ...
****************************************

HyperScale constraints report

   Constraint: data_arrival

   Instance   Pin(Port name)        Window type   CLK           Block  Top   Slack
   -------------------------------------------------------------------------------
   mtd_core   mtd_core/u0/A(clk1)   max_rise      top_CLK1(r)   N/A    0.23  N/A
   mtd_core   mtd_core/u0/A(clk1)   min_rise      top_CLK1(r)   N/A    0.23  N/A
   core3      core3/u0/A(clk1)      min_fall      top_CLK1(f)   N/A    0.23  N/A

   Constraint: clock_latency

   Instance   Pin(Port name)        Window type   CLK           Block  Top   Slack
   -------------------------------------------------------------------------------
   mtd_core   mtd_core/u0/A(clk1)   max_rise      top_CLK1(r)   0.00   0.66  -0.66
   mtd_core   mtd_core/u0/A(clk1)   max_fall      top_CLK1(r)   N/A    0.66  N/A
   core3      core3/u0/A(clk1)      min_fall      top_CLK1(r)   N/A    0.66  N/A

   Constraint: input_slews

   Instance   Pin(rise/fall)        Window type   CLK           Block  Top   Slack
   -------------------------------------------------------------------------------
   core       core/in1(rise)        max_rise      ---           N/A    0.03  N/A
   core3      core3/in1(rise)       max_rise      ---           N/A    0.03  N/A

1
```

SEE ALSO

report_constraint(2)
boundary_logic_value_violations(3)
clock_latency_violations(3)
clock_skew_with_uncertainty_violations(3)
data_arrival_violations(3)
input_slews_violations(3)
non_auto_fixable_violations(3)

auto_link_disable

Disables the autolink process.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **false** (the default), many PrimeTime commands attempt to automatically link the current design for you. For example, the **set_load** command 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 autolink, so it is not attempted.

Setting the **auto_link_disable** variable to **true** disables the autolink 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 the **auto_link_disable** variable to **true**.
3. Source the script.
4. Reset the **auto_link_disable** variable to **false**.

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

SEE ALSO

link_design(2)

auto_wire_load_selection

Enables the automatic selection of wire load models.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Setting this variable to **true** (the default) enables the automatic selection of wire load models, which are used to estimate net capacitances and resistances from the net fanout. When you set this variable to **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 (using the **set_wire_load_model** command), automatic wire load selection for that block is disabled.

SEE ALSO

`report_wire_load(2)`

```
set_wire_load_min_block_size(2)
set_wire_load_selection_group(2)
```

boundary_ideal_network_violations

This man page describes **boundary_ideal_network** hierarchical boundary check violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **boundary_ideal_network** violation indicates a mismatch in the ideal networks between the top- and block-level runs at this object.

WHAT NEXT

This is a **non_auto_fixable** violation. To fix a **boundary_ideal_network** violation, use the **set_ideal_network** or **remove_ideal_network** commands on the object at the top- or block-level runs.

SEE ALSO

```
remove_ideal_network(2)
report_constraint(2)
set_ideal_network(2)
non_auto_fixable_violations(3)
```

boundary_logic_value_violations

This man page describes **boundary_logic_value** hierarchical boundary check violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **boundary_logic_value** violation indicates that the tool detected a conflict in the logic values set on or propagated to the specified hierarchical boundary pin or port.

For example, a **boundary_logic_value** violation occurs under the following conditions:

- HyperScale block-level analysis sets a SCANEN port with a case value of 0 and performs timing analysis.
- When the block is instantiated at the top level, a logic value not equal to 0 (either 1 or no case or constant values at all) propagates through the hierarchical pin.

Note:
In the context of a **boundary_logic_value** HyperScale violation, the reported logic value might not always come from a user-defined **set_case_analysis**; it can also arise from a functional constant defined in the design, such as from Verilog, or from library cells with tie-high or tie-low connections.

The HyperScale analysis flow requires the constraints to be consistent between the block- and top-level runs. This include the same settings for case values and design constants on the block boundary. The HyperScale top flow captures the logic values for instance boundary and automatically uses the captured value to override the user-defined settings at block-level analysis to improve consistency of case value settings on ports. Therefore, this type of violation is considered auto-fixable by PrimeTime runs.

In multi-instance blocks where the HyperScale top-level run automatically merges into a common context for a single block-level analysis, the tool resolves conflicts in case analysis values on the same port across instances by setting no case values on the port for block-level analysis. In this situation, HyperScale does not report a **boundary_logic_value** violation for different instances because they are regarded as expected.

WHAT NEXT

This is an **auto_fixable** violation. Confirm the case value difference, and either

- Manually align the constraints

- Apply the latest top context in the next block-level run to force it to align with the top-level context.

EXAMPLES

The following example shows a summary report for a **boundary_logic_value** violation.

```
pt_shell> report_constraint -boundary_check_include {boundary_logic_value} \
          -all_violators

****************************************
Report : constraint
         -all_violators
         -boundary_check
 ...
****************************************

HyperScale constraints report

   Constraint: boundary_logic_value

   Instance        Num_vio        Reason
   -------------------------------------------------
   block           1              top/block mismatch
```

The following example shows a verbose report:

```
pt_shell> report_constraint -boundary_check_include {boundary_logic_value} \
          -all_violators -verbose

****************************************
Report : constraint
         -verbose
 ...
****************************************

HyperScale constraints report

   Constraint: boundary_logic_value

   Instance      Pin              Attributes       Block     Top
   ---------------------------------------------------------------
   block         core/TSTEN       Case Value       1         X
```

SEE ALSO

```
get_attribute(2)
report_case_analysis(2)
report_constraint(2)
auto_fixable_violations(3)
```

bus_naming_style

Sets the naming format for a specific element of a bus.

TYPE

string

DEFAULT

%s[%d]

DESCRIPTION

This 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 appear in the application.

The default is "%s[%d]". For example, for bus A and index 12, the name would be A[12].

SEE ALSO

`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

This variable specifies the name of a log file to be generated during propagation of constant values from case analysis or from nets tied to logic 0 or to logic 1.

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.

If the file name is specified with the .gz extension, the output file is written in compressed (gzip) format.

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

Specifies whether case analysis is propagated through integrated clock-gating cells.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, this variable is set to **false**, which means that the tool does not propagate logic constants through integrated clock-gating cells. Regardless of whether the integrated clock-gating cell is enabled or disabled, no logic values propagate in the fanout of the cell.

If you set this variable to **true**, the tool propagates logic constants through integrated clock-gating cells. Set this variable before using the **update_timing** command.

An integrated clock-gating cell is enabled when its enable pin (or test enable pin) is set to logic 1. If the cell is disabled, the disable logic value for the cell is propagated in its fanout. For example, when the latch_posedge integrated clock gating is disabled, it propagates a logic 0 in its fanout.

Integrated clock-gating cells have the Library Compiler **clock_gating_integrated_cell** attribute. PrimeTime supports integrated clock-gating cells with latch memory elements. These cells are sequential in nature.

Integrated clock-gating cells that have no memory element are not sequential cells. Therefore, they are considered combinational and always propagate.

SEE ALSO

```
remove_case_analysis(2)
set_case_analysis(2)
```

case_analysis_sequential_propagation

Specifies whether case analysis is propagated across sequential cells.

TYPE

string

DEFAULT

never

DESCRIPTION

This variable specifies 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. When the **case_analysis_propagate_through_icg** variable is set to true, these types of integrated clock gating cells only propagate logic values.

SEE ALSO

```
remove_case_sequential_propagation(2)
set_case_analysis(2)
set_case_sequential_propagation(2)
case_analysis_propagate_through_icg(3)
```

category_node_attributes

Describes the predefined application attributes for category_node objects.

DESCRIPTION

name

Type: string
Returns the name of the category_node object.

object_class

Type: string
Returns the class of the object, which is a constant equal to category_node. You cannot set this attribute.

SEE ALSO

get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)

category_tree_attributes

Describes the predefined application attributes for category_tree objects.

DESCRIPTION

name

Type: string
Returns the name of the category_tree object.

object_class

Type: string
Returns the class of the object, which is a constant equal to category_tree. You cannot set this attribute.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

cell_attributes

Describes the predefined application attributes for cell objects.

DESCRIPTION

area

Type: float
Returns the area of the cell. If the cell is hierarchical, this includes net area.

base_name

Type: string
Returns the leaf name of the cell. For example, the base_name of cell U1/U2/U3 is U3.

block_config_name

Type: string
Returns the named session for saved HyperScale block data; applies to hierarchical cells for HyperScale top-level runs.

block_config_path

Type: string
Returns the configuration path of the HyperScale block; applies to hierarchical cells for HyperScale top-level runs.

bottleneck_cost

Type: float
Returns the bottleneck cost computed by the report_bottleneck command; requires running this command before querying the attribute.

clock_pin_power

Type: double
Returns the internal power of the cell's clock pins that have the is_clock_pin library attribute set to true. If there are multiple clock pins on the cell, the attribute returns the sum of the internal power on the clock pins. The attribute value does not include power associated with the internal clock pins of the cell.

critical_path_max

Type: string
Returns a structured list that contains the name of the path group, levels of logic of the critical path, length, and setup slack for each path group present at the block level.

critical_path_min

Type: string
Returns a structured list containing the name of the path group, levels of logic of the critical path, length, and hold slack for each path group present at the block level.

disable_timing

Type: boolean
Returns true if the timing for the cell has been marked as disabled in set_disable_timing. You can set and unset the disable_timing attribute.

dont_touch

Type: boolean
Returns true if the cell is protected from change by the set_dont_touch command. When this attribute is set to true, the cell cannot be resized, removed, or replaced by the ECO fixing commands (fix_eco_drc, fix_eco_power, and fix_eco_timing). This attribute exists for a cell only after the attribute is created by the set_dont_touch command.

drc_violations

Type: string
Returns a list containing, for each violated DRC check type: the check name, the number of violations, and the violation cost.

dynamic_power

Type: double
Returns the dynamic power of the cell in watts. It is the sum of the internal power and switching power.

early_fall_cell_check_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_fall_clk_cell_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_fall_clk_net_delta_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_fall_clk_net_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_fall_data_cell_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_fall_data_net_delta_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_fall_data_net_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_rise_cell_check_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_rise_clk_cell_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_rise_clk_net_delta_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_rise_clk_net_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_rise_data_cell_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_rise_data_net_delta_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

early_rise_data_net_derate_factor

Type: float
Returns the early timing derating factor, set by the set_timing_derate command, that applies to the cell.

escaped_full_name

Type: string
Returns the name of the cell. Any literal hierarchy characters are escaped with a backslash.

full_name

Type: string
Returns the complete name of the cell. For example, the full name cell U3 within cell U2 within cell U1 is U1/U2/U3. The full_name attribute is not affected by the current_instance setting.

gate_leakage_power

Type: double
Returns the gate leakage power of the cell in watts. Gate leakage power is the leakage power from the source to the gate or the gate to the drain.

glitch_power

Type: double
Returns the glitch power of the cell in watts. Glitch power is considered part of the dynamic power.

has_multi_ground_rails

Type: boolean
Returns true if the cell has multiple ground rails.

has_multi_power_rails

Type: boolean
Returns true if the cell has multiple power rails.

has_rail_specific_power_tables

Type: boolean
Returns true if the cell has power tables attached to rails.

internal_power

Type: double
Returns the internal power of the cell in watts. Internal power is any dynamic power dissipated within

the boundary of the cell.

internal_power_derate_factor

Type: float
Returns the power derating factor on the cell. To set this value, use the set_power_derate command.

intrinsic_leakage_power

Type: double
Returns the intrinsic leakage power of the cell in watts. Most intrinsic leakage power results from source-to-drain subthreshold leakage.

is_black_box

Type: boolean
Returns true if the cell's reference is not linked to a library cell or design. This attribute is read-only.

is_case_sequential_propagation

Type: boolean
Returns true if the cell is enabled for sequential case propagation by the set_case_sequential_propagation command.

is_clock_gating_check

Type: boolean
Returns true if the cell is a clock-gating check cell.

is_clock_network_cell

Type: boolean
Returns true if the cell is in the clock network of any clock.

is_combinational

Type: boolean
Returns true if the corresponding library cell has no sequential timing arcs. This attribute is not valid for hierarchical and black-box cells. See also the is_hierarchical and is_black_box cell attributes.

is_design_mismatch

Type: boolean
Returns true if the cell has a mismatch between the block and the top-level design.

is_edited

Type: boolean
Returns true if the hierarchical cell has been uniquified as a result of a netlist editing (ECO) change. It is only defined for hierarchical cells.

is_fall_edge_triggered

Type: boolean
Returns true if the cell is used as a falling-edge-triggered flip-flop.

is_hierarchical

Type: boolean
Returns true for hierarchical cells and false for leaf cells. Hierarchical cells represent instances of other designs, while leaf cells represent instances of library cells.

is_hyperscale_block

Type: boolean
Returns true if the hierarchical cell is a HyperScale block.

is_ideal

Type: boolean
Returns true if the cell has been marked ideal using the set_ideal_network command.

is_integrated_clock_gating_cell

Type: boolean
Returns true if the cell is defined in the library as an integrated clock-gating cell.

is_interface_logic_model

Type: boolean
Returns true if the cell is an interface logic model. PrimeTime automatically sets this attribute to true when you create an interface logic model with the create_ilm command. The attribute allows the check_timing command to identify interface logic models and suppress false reporting of unconnected clock pins.

is_macro_switch

Type: boolean
Returns true if the reference library cell is defined as a fine-grain switch cell in the library.

is_memory_cell

Type: boolean
Returns true if the cell is a memory cell. This attribute is inherited from the reference library cell. It is used by PrimePower to identify memory cells when reporting power or activity annotation for different cell types, such as sequential, combinational, and memory cell types.

is_mim_context_reference

Type: boolean
Returns true if the cell is used as the reference instance during hyperscale multi-instance-merging (MIM) flow. It is used (1) to provide a mechanism to query and script based on tool auto picked reference instance and (2) to provide confirmation on user specified reference instance.

is_mux

Type: boolean
Returns true if the cell is a multiplexer.

is_negative_level_sensitive

Type: boolean
Returns true if the cell is used as a negative level-sensitive latch.

is_pad_cell

Type: boolean
Returns true if the cell is a pad cell.

is_positive_level_sensitive

Type: boolean
Returns true if the cell is used as a positive level-sensitive latch.

is_power_standby_cell

Type: boolean
Returns true if the cell is a power standby cell.

is_rise_edge_triggered

Type: boolean
Returns true if the cell is used as a rising-edge-triggered flip-flop.

is_sequential

Type: boolean
Returns true if the corresponding library cell has at least one sequential timing arc. This attribute is not valid for hierarchical and black-box cells. See also the is_hierarchical and is_black_box cell attributes.

is_three_state

Type: boolean
Returns true if the cell is a three-state device.

late_fall_cell_check_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_fall_clk_cell_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_fall_clk_net_delta_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_fall_clk_net_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_fall_data_cell_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_fall_data_net_delta_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_fall_data_net_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_rise_cell_check_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_rise_clk_cell_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_rise_clk_net_delta_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_rise_clk_net_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_rise_data_cell_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_rise_data_net_delta_derate_factor

Type: float
Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

late_rise_data_net_derate_factor

Type: float

Returns the late timing derating factor, set by the set_timing_derate command, that applies to the cell.

leaf_cell_count

Type: integer
Returns the cell count in the hierarchical block that was used to create the abstracted hierarchical instance.

leakage_power

Type: double
Returns the leakage power of the cell in watts. It is the power that the cell dissipates when it is not switching. Leakage power is the sum of the intrinsic leakage and gate leakage.

leakage_power_derate_factor

Type: float
Returns the power derating factors, specified using the set_power_derate command, that apply to the cell.

lib_cell

Type: collection
Returns a collection of library cells for this cell; this attribute is defined only on leaf cells.

number_of_pins

Type: integer
Returns the number of pins on the cell. The number of pins can be different before and after linking. For example, if some pins were unconnected in a Verilog instance, after linking to the lower-level design, additional pins can be created on the cell.

object_class

Type: string
Returns the class of the object, which is "cell". You cannot set this attribute.

original_ref_name

Type: string
Returns the original reference name of hierarchical blocks within the HyperScale model. This attribute applies to hierarchical cells for HyperScale top-level runs. This attribute is needed to apply constraints and LEF files to non-HyperScale subdesigns within HyperScale models at the top level.

parent_cell

Type: collection
Returns the parent hierarchical cell of the cell.

peak_power

Type: double
Returns the peak power of the cell in watts. This is the largest power value for that cell in the simulation

waveform.

peak_power_end_time

Type: double
Returns the end time of the time interval in which peak power is measured for that cell. The unit of measurement is nanoseconds.

peak_power_start_time

Type: double
Returns the start time of the time interval in which peak power is measured for that cell. The unit of measurement is nanoseconds.

pg_pin_info

Type: collection
Returns a collection of these pg_pin_info objects: pin_name, type, voltage_for_max_delay, voltage_for_min_delay, supply_connection.

pin_count

Type: integer
Returns the pin count in the hierarchical block that was used to create the abstracted hierarchical instance.

power_cell_type

Type: string
Returns the predefined power group to which the cell belongs. Use the set_user_attribute command to set the attribute to one of these predefined power groups: clock_network, register, combinational, sequential, memory, io_pad, or black_box. You cannot use this attribute to assign a cell to a user-defined power group. The power_cell_type cell attribute overrides the power_cell_type library cell attribute. PrimePower uses this attribute in both averaged and time-based power analysis modes.

power_states

Type: double
Returns the number of power state changes for the cell. This is the sum of the number of transitions on all input and output pins of the cell.

ref_name

Type: string
Returns the name of the design or library cell of which the cell is (or will be) an instantiation; also known as the reference name. The linker looks for a design or library cell by this name to resolve the reference.

switching_power

Type: double
Returns the switching power of the cell in watts. It is the power dissipated by the charging and discharging of the load capacitance at the output of the cell.

switching_power_derate_factor

Type: float
Returns the power derating factor, specified by the set_power_derate command, that applies to the cell.

temperature_max

Type: float
Returns the maximum (worst) case temperature for the cell specified by the operating condition or the set_temperature command.

temperature_min

Type: float
Returns the minimum (best) case temperature for the cell specified by the operating condition or the set_temperature command.

timing_model_type

Type: string
Returns the timing model type of the cell. The possible values are ITS (interface timing specification), quick timing model, extracted, and none (normal library model). You can set this attribute.

total_power

Type: double
Returns the total power of the cell in watts. It is the sum of dynamic power and leakage power.

upf_isolation_strategy

Type: string
Returns the strategy name (created by the set_isolation command) that was used to derive rail supply and ground nets of the cell.

upf_retention_strategy

Type: string
Returns the strategy name (created by the set_retention command) that was used to derive backup rail supply and ground nets.

wire_load_model_max

Type: string
Returns the name of the wire load model effective on a hierarchical cell for the maximum operating condition. You can set this attribute.

wire_load_model_min

Type: string
Returns the name of the wire load model effective on a hierarchical cell for the minimum operating condition (valid in on-chip variation analysis). You can set this attribute.

wire_load_selection_group_max

Type: string
Returns the name of the wire load selection group on a hierarchical cell for the maximum operating condition. You can set this attribute.

wire_load_selection_group_min

Type: string
Returns the name of the wire load selection group on a hierarchical cell for the minimum operating condition. You can set this attribute.

x_coordinate_max

Type: float
Returns the maximum x-coordinate of the area occupied by the cell.

x_coordinate_min

Type: float
Returns the minimum x-coordinate of the area occupied by the cell.

x_transition_power

Type: double
Returns the dynamic power used by the cell during transitions from the unknown (X) state to a known state (0 or 1).

y_coordinate_max

Type: float
Returns the maximum y-coordinate of the area occupied by the cell.

y_coordinate_min

Type: float
Returns the minimum y-coordinate of the area occupied by the cell.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

clock_attributes

Describes the predefined application attributes for clock objects.

DESCRIPTION

clock_latency_fall_max

Type: float
Returns the maximum fall latency (insertion delay) set by the set_clock_latency command.

clock_latency_fall_min

Type: float
Returns the minimum fall latency (insertion delay) set by the set_clock_latency command.

clock_latency_rise_max

Type: float
Returns the maximum rise latency (insertion delay) set by the set_clock_latency command.

clock_latency_rise_min

Type: float
Returns the minimum rise latency (insertion delay) set by the set_clock_latency command.

clock_network_pins

Type: collection
Returns a collection of pin and port objects in the propagation path of the clock.

clock_source_latency_early_fall_max

Type: float
Returns the maximum early falling source latency set by the set_clock_latency command.

clock_source_latency_early_fall_min

Type: float
Returns the minimum early falling source latency set by the set_clock_latency command.

clock_source_latency_early_rise_max

Type: float
Returns the maximum early rising source latency set by the set_clock_latency command.

clock_source_latency_early_rise_min

Type: float
Returns the minimum early rising source latency set by the set_clock_latency command.

clock_source_latency_late_fall_max

Type: float
Returns the maximum late falling source latency set by the set_clock_latency command.

clock_source_latency_late_fall_min

Type: float
Returns the minimum late falling source latency set by the set_clock_latency command.

clock_source_latency_late_rise_max

Type: float
Returns the maximum late rising source latency set by the set_clock_latency command.

clock_source_latency_late_rise_min

Type: float
Returns the minimum late rising source latency set by the set_clock_latency command.

clock_source_latency_pins

Type: collection
Returns a collection of pins and ports in the source latency network of a generated clock. This attribute is undefined for master clocks.

full_name

Type: string
Returns the name of the clock. This is set with the create_clock command. It is either the name given with the -name option, or the name of the first object to which the clock is attached. After the name is set, this attribute is read-only.

generated_clocks

Type: collection
Returns a collection of generated clock objects for which the clock is the source. This attribute is defined for any clock that is the parent source of one or more generated clocks.

hold_uncertainty

Type: float
Returns the clock uncertainty (skew) of the clock used for hold (and other minimum delay) timing checks, as set by the set_clock_uncertainty command.

is_active

Type: boolean
Returns true if the clock is active, the default state. To set clocks as active or inactive, use the set_active_clocks command.

is_generated

Type: boolean
Returns true for a generated clock. To create a generated clock, use the create_generated_clock command.

master_clock

Type: collection
Returns the master clock of a generated clock. This attribute is defined only for generated clocks.

master_pin

Type: collection
Returns the master source pin or port object used to determine the identity and polarity of the master clock. This corresponds to the pin or port specified by the -source option of the create_generated_clock command. This attribute is defined for generated clocks only.

max_capacitance_clock_path_fall

Type: float
Returns the upper limit for the falling maximum capacitance for all pins in the clock path, as set by the set_max_capacitance command.

max_capacitance_clock_path_rise

Type: float
Returns the upper limit for the rising maximum capacitance for all pins in the clock path, as set by the set_max_capacitance command.

max_capacitance_data_path_fall

Type: float
Returns the upper limit for the falling maximum capacitance for all pins in the data path launched by the clock, as set by the set_max_capacitance command.

max_capacitance_data_path_rise

Type: float
Returns the upper limit for the rising maximum capacitance for all pins in the data path launched by the clock, as set by the set_max_capacitance command.

max_fall_delay

Type: float
Returns the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects, as set by the set_max_delay command.

max_rise_delay

Type: float
Returns the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects, as set by the set_max_delay command.

max_time_borrow

Type: float
Returns the upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches, as set by the set_max_time_borrow command. Units are those of the logic library.

max_transition_clock_path_fall

Type: float
Returns the upper limit for the falling maximum transition for all pins in the clock path, as set by the set_max_transition command.

max_transition_clock_path_rise

Type: float
Returns the upper limit for the rising maximum transition for all pins in the clock path, as set by the set_max_transition command.

max_transition_data_path_fall

Type: float
Returns the upper limit for the falling maximum transition for all pins in the data path launched by the clock, as set by the set_max_transition command.

max_transition_data_path_rise

Type: float
Returns the upper limit for the rising maximum transition for all pins in the data path launched by the clock, as set by the set_max_transition command.

min_fall_delay

Type: float
Returns the minimum falling delay on ports, clocks, pins, cells, or paths between such objects, as set by the set_min_delay command.

min_rise_delay

Type: float
Returns the minimum rising delay on ports, clocks, pins, cells, or paths between such objects, as set by the set_min_delay command.

object_class

Type: string
Returns the class of the object, which is "clock". You cannot set this attribute.

period

Type: float
Returns the clock period (or cycle time), which is the shortest time during which the clock waveform repeats. For a simple waveform with one rising and one falling edge, the period is the difference between successive rising edges. It is set by the create_clock -period command.

propagated_clock

Type: boolean
Returns true if clock latency (insertion delay) is determined by propagating delays from the clock source to destination register clock pins, as set by the set_propagated_clock command. If this attribute is not present, ideal clocking is assumed.

setup_uncertainty

Type: float
Returns the clock uncertainty (skew) of the clock used for setup (and other maximum delay) timing checks, as set by the set_clock_uncertainty command.

sources

Type: collection
Returns a collection of the source pins or ports of the clock, as defined by the create_clock command.

waveform

Type: string
Returns a string representation of the clock waveform, as defined by the create_clock command. For example, a clock rising at 2.5 and falling at 5.0 has a waveform attribute value of {2.5 5.0}.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

clock_attributes_violations

This man page describes **clock_attributes** violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **clock_attributes** violation indicates that a clock defined at the block level is successfully mapped to a clock at the top level, but certain elements of the clock definition do not match.

The following clock attributes are checked:

- **is_active** - true if the clock is active.
- **is_propagated** - true if the clock is propagated.
- **sources** - List of source pins or ports of the clock.
- **edges** - List of clock edges.
- **clock_sense** - Clock sense.

Clocks can be made active or inactive with the **set_active_clocks** command. The **is_propagated** attribute is **true** if the clock latency is determined by propagating delays from clock sources to register clock pins. Otherwise, ideal clocking is assumed. Set with **set_propagated_clock**.

WHAT NEXT

A **clock_attributes** violation is a **non_auto_fixable** violation. To resolve this violation, change the definition of clock in constraints based on the information given by the **report_constraint -verbose** command.

EXAMPLES

The following report summarizes clock attribute mismatch violations.

```
pt_shell> report_constraint -boundary_check_include {clock_attributes} -all_violators

****************************************
Report : constraint
        -all_violators
        -boundary_check
Design : wrapper
 ...
****************************************

HyperScale constraints report


   Constraint: clock_attributes

   Instance               Num_vio        Reason
   ----------------------------------------------------------------
   core1                  1              non-propagated block clock
   core3                  1              clock sources mismatch
```

The following example shows a verbose report. In this example, not propagated block-level clock CLK is mapped to a propagated top-level clock top_CLK. Block-level clock CLK has only one source in the instance core3, but the top-level clock top_CLK has two sources.

```
pt_shell> report_constraint -boundary_check_include {clock_attributes} \
          -all_violators -verbose

****************************************
Report : constraint
        -all_violators
        -verbose
        -boundary_check
Design : wrapper
 ...
****************************************

HyperScale constraints report

   Constraint: clock_attributes

   Instance    Attribute name      Block level (clock)    Top level (clock)
   ------------------------------------------------------------------------------
   core1       is_propagated       false (CLK)            true (top_CLK)
   core3       sources             {core3/clk1} (CLK)     {core3/clk1,core3/clk2} (top_CLK)
```

SEE ALSO

get_attribute(2)
report_clock(2)
report_constraint(2)
set_active_clocks(2)
non_auto_fixable_violations(3)

clock_latency_violations

This man page describes **clock_latency** hierarchical boundary check violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **clock_latency** violation indicates a mismatch of clock latency at a given pin between the top- and block-level analysis.

WHAT NEXT

This is an **auto_fixable** violation. This violation can be automatically fixed in subsequent iterations, but the fix is not guaranteed because of interactions between blocks. Also, fix any clock existence violations first; otherwise, it is possible that arrival violations will not be fixed.

EXAMPLES

The following verbose report shows violations in clock latency settings applied at block-level analysis that are different to the top-level settings.

```
pt_shell> report_constraint -boundary_check_include {clock_latency} \
          -all_violators -verbose

****************************************
Report : constraint
         -all_violators
         -verbose
         -boundary_check
Design : top
 ...
****************************************

HyperScale constraints report

   Constraint: clock_latency
```

```
Instance   Pin(Port name)     Window type   CLK          Block   Top     Slack
-------------------------------------------------------------------------------
core       core/u0/A(clk1A)   max_fall      top_CLK(f)   0.320   0.330   -0.010
core       core/u0/A(clk1A)   max_rise      top_CLK(r)   0.420   0.440   -0.020
```

SEE ALSO

report_constraint(2)
set_clock_latency(2)
auto_fixable_violations(3)

clock_mapping_violations

This man page describes **clock_mapping** violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **clock_mapping** violation indicates that PrimeTime cannot find a matching clock for the reported block or top-level clock with the given attributes.

A clean and complete mapping between top and block-level clocks must be established to perform valid timing analysis in HyperScale flow.

The tool automatically performs clock mapping according to the following characteristics captured for block and top-level clocks (in order of highest to lowest priority):

- Overlapping clock source objects on the same physical source network
- Matching clock periods
- Matching clock waveforms
- Matching clock names as prefix

For HyperScale analysis, you must provide complete and consistent definitions for all clocks used in both block- and top-level analysis. At the top level, the full-chip flat constraints are often used with all clocks defined for the entire design, including HyperScale subblocks; this implies that the block constraints supplied for HyperScale block-level analysis must consistently define all clocks relevant to the given block. Clock mapping is the step to check and establishes this clock consistency between top- and block-level analysis. It is performed at both the block and top level. Any irregularity is reported as a **clock_mapping** violation.

Typical reasons for **clock_mapping** violations include:

- extra_clock - It means that an extra and redundant clock is defined at block level analysis with no corresponding clock at top level.
- missing_clock - It means that a top-level clock propagating to a port on the boundary of a HyperScale block, but the constraints used for the block level analysis for that block did not define a clock with matching characteristics.

WHAT NEXT

This is a **non_auto_fixable** violation. To validate and fix the violation, edit block- or top-level timing constraints to ensure complete and consistent definitions of all clocks between top and block level analysis.

EXAMPLES

The following summary report shows clock mapping violations.

```
pt_shell> report_constraint -boundary_check_include {clock_mapping} -all_violators

****************************************
Report : constraint
        -all_violators
        -boundary_check
Design : wrapper
 ...
****************************************

HyperScale constraints report

   Constraint: clock_mapping

   Instance                Num_vio        Reason
   ----------------------------------------------------------
   core1                   1              missing_clock
   core2                   1              missing_clock
```

The following verbose report shows all clock mapping violations where clocks are defined at a block level, but no corresponding top-level clocks can be matched to these block-level clocks.

```
pt_shell> report_constraint -boundary_check_include {clock_mapping} \
          -all_violators -verbose

****************************************
Report : constraint
        -all_violators
        -verbose
        -boundary_check
Design : wrapper
 ...
****************************************

HyperScale constraints report

Constraint: clock_mapping

Instance   Block level clock   Top level Clock   Attributes                    Block clock ref_pin
-------------------------------------------------------------------------------------------------
core1      CLK_extra1          ---               p=16.00,e={0.00 8.00},a={p A}  {core1/clk1}
core2      CLK_extra1          ---               p=16.00,e={0.00 8.00},a={p A}  {core2/clk1}
```

In the preceding example, the attribute column shows characteristics for the clock named in the column 'Block level'. These characteristics are the period (denoted by the character 'p'), clock edges ('e'), and other attributes ('a'). The possible values for other attributes are: "A" means that the clock is active, "p" means that the clock is propagated.

The following verbose report shows all clock mapping violations where a top level clock has propagated to boundary ports of block instances core1 and core2, but no block-level clock is using these ports as clock sources.

```
pt_shell> report_constraint -boundary_check_include {clock_mapping} \
          -all_violators -verbose

****************************************
Report : constraint
        -all_violators
        -verbose
        -boundary_check
Design : wrapper
 ...
****************************************

HyperScale constraints report

Constraint: clock_mapping

Instance  Block level clock   Top level Clock   Attributes                Block clock ref_pin
-------------------------------------------------------------------------------------------------
core1     ---                 CLKx              p=12.00,e={0 8},a={p A}    {core1/clk2 core1/clk1}
core2     ---                 CLKx              p=12.00,e={0 8},a={p A}    {core2/clk2 core2/clk1}
```

To fix violations shown in the preceding example, create block-level clocks using the list of source pins printed in the last column of the report.

SEE ALSO

report_clock(2)
report_constraint(2)
non_auto_fixable_violations(3)

clock_relations_violations

This man page describes **clock_relations** violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **clock_relations** violation means that there is a mismatch of asynchronous or exclusive relation between clocks at top and block levels.

WHAT NEXT

This is a **non_auto_fixable** violation. Use the **set_clock_groups** command to make relations between clocks the same at the top and block levels of the analysis.

EXAMPLES

The following example shows a summary report for clock_relations violations.

```
pt_shell> report_constraint -boundary_checks {clock_relations} -all_violators

****************************************
Report : constraint
        -all_violators
        -boundary_checks
Design : wrapper
 ...
****************************************

HyperScale constraints report

   Constraint: clock_relations

   Instance             Num_vio        Reason
   ------------------------------------------------------------
   core2                1              clock async mismatch
   core                 2              clock async mismatch
```

The following example shows a verbose report for clock_relations.

```
pt_shell> report_constraint -boundary {clock_relations} -all_violators -verbose

****************************************
Report : constraint
        -all_violators
        -verbose
        -boundary_checks
Design : wrapper
 ...
****************************************

HyperScale constraints report

   Constraint: clock_relations

   Instance   Clock relation   Block level            Top level
   ------------------------------------------------------------------------
   core2      async            {int_CLK} <-> {top_CLK} {core2/int_CLK} <-> {}
   core       async            {int_CLK} <-> {top_CLK} {core/int_CLK} <-> {}
```

SEE ALSO

```
report_clock(2)
report_constraint(2)
set_clock_groups(2)
non_auto_fixable_violations(3)
```

clock_skew_with_uncertainty_violations

This man page describes **clock_skew_with_uncertainty** violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **clock_skew_with_uncertainty** violation indicates that a clock defined at block level is successfully mapped to a clock at the top level, but the interclock skew values associated with these two clocks are different.

WHAT NEXT

This is an **auto_fixable** violation. If a **clock_skew_with_uncertainty** violation is due to a mismatch of clock latency, it is automatically fixed in a subsequent iteration; however, the fix is not guaranteed because of interactions between blocks. Also, you must first fix any clock existence violations. If the violations are due to mismatch of uncertainty, change the clock uncertainty for the block- or top-level clocks with the **set_clock_uncertainty** command.

EXAMPLES

The following summary report shows **clock_skew_with_uncertainty** violations.

```
pt_shell> report_constraint -boundary_check_include {clock_skew_with_uncertainty} \
          -all_violators

****************************************
Report : constraint
        -all_violators
        -boundary_check
Design : wrapper
 ...
****************************************

HyperScale constraints report

   Constraint: clock_skew_with_uncertainty
```

```
   Instance          Num_vio        Reason
   -------------------------------------------------
   core3             2              top/block mismatch
   core1             2              top/block mismatch
```

The following verbose report shows all **clock_skew_with_uncertainty** violations.

```
pt_shell> report_constraint -boundary_check_include {clock_skew_with_uncertainty} \
          -all_violators -verbose

****************************************
Report : constraint
        -all_violators
        -verbose
        -boundary_check
Design : wrapper
 ...
****************************************

HyperScale constraints report

   Constraint: clock_skew_with_uncertainty

   Instance   Type    From Clock    To Clock        Block   Top    Slack
   ---------------------------------------------------------------------
   core3      setup   CLK3 (fall)   CLK4_v (rise)   0.00    4.00   -4.00
   core3      hold    CLK1 (rise)   CLK2_v (fall)   0.00    1.00   -1.00
   core1      setup   CLK3 (fall)   CLK4_v (rise)   0.00    4.00   -4.00
   core1      hold    CLK1 (rise)   CLK2_v (fall)   0.00    1.00   -1.00
```

SEE ALSO

report_clock(2)
report_constraint(2)
set_clock_latency(2)
set_clock_uncertainty(2)
auto_fixable_violations(3)

clock_uncertainty_violations

This man page describes **clock_uncertainty** violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **clock_uncertainty** violation indicates that a clock defined at block level is successfully mapped to a clock at top level, but the uncertainty (skew) values associated with these two clocks are different.

WHAT NEXT

This is a **non_auto_fixable** violation. To resolve this violation, change the clock uncertainty for the block- or top-level clocks with the **set_clock_uncertainty** command.

EXAMPLES

The following summary report shows **clock_uncertainty** violations.

```
pt_shell> report_constraint -boundary_check_include {clock_uncertainty} -all_violators

****************************************
Report : constraint
        -all_violators
        -boundary_check
Design : wrapper
 ...
****************************************

HyperScale constraints report

   Constraint: clock_uncertainty

   Instance          Num_vio        Reason
   -------------------------------------------------
   core3             2              top/block mismatch
   core1             2              top/block mismatch
```

The following verbose report shows all clock uncertainty mismatch violations.

```
pt_shell> report_constraint -boundary_check_include {clock_uncertainty} \
          -all_violators -verbose

****************************************
Report : constraint
        -all_violators
        -verbose
        -boundary_check
Design : wrapper
 ...
****************************************

HyperScale constraints report

   Constraint: clock_uncertainty

   Instance   Type    From Clock    To Clock        Block   Top    Slack
   ---------------------------------------------------------------------
   core3      setup   CLK3 (fall)   CLK4_v (rise)   0.00    4.00   -4.00
   core3      hold    CLK1 (rise)   CLK2_v (fall)   0.00    1.00   -1.00
   core1      setup   CLK3 (fall)   CLK4_v (rise)   0.00    4.00   -4.00
   core1      hold    CLK1 (rise)   CLK2_v (fall)   0.00    1.00   -1.00
```

SEE ALSO

```
report_clock(2)
report_constraint(2)
set_clock_uncertainty(2)
non_auto_fixable_violations(3)
```

collection_result_display_limit

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

TYPE

integer

DEFAULT

100

DESCRIPTION

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

When a command, such as the **add_to_collection**) command, is issued at the command prompt, the result is implicitly queried, as though the **query_objects** command was called. To limit the number of objects displayed, set the **collection_result_display_limit** variable to an appropriate integer.

A value of -1 displays all objects; a value of 0 displays the collection handle ID instead of the object names in the collection.

SEE ALSO

collections(2)
query_objects(2)

correlation_attributes

Describes the predefined application attributes for correlation objects.

DESCRIPTION

full_name

Type: string
Returns the full name of the correlation object.

object_class

Type: string
Returns the class of the object, which is a constant equal to correlation. You cannot set this attribute.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

create_clock_no_input_delay

Specifies delay propagation characteristics of clock sources created using the **create_clock** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable affects the delay propagation characteristics of clock sources created using the **create_clock** command. When the variable is set to **false** (the default), the clock sources used in the data path are established as timing startpoints. The clock sources in the design propagates rising delays on every rising clock edge and propagates falling delays on every falling clock edge. To disable this behavior, set the **create_clock_no_input_delay** variable to **true**.

SEE ALSO

`create_clock(2)`

data_arrival_violations

This man page describes **data_arrival** violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **data_arrival** violation reported in HyperScale top-level analysis means that the data arrival windows defined at the block-level run do not fully contain the actual arrival windows propagated at the same pin at the top level due to one of the following conditions:

- The minimum or early window analyzed at the block level is larger or later than the actual minimum arrivals reaching the specified pin or port.
- The maximum or late window at the block level is smaller or earlier than the actual maximum arrivals at the top level.

This HyperScale boundary constraint violation implies potential timing violations within the internal register-to-register paths of the block because those paths are not visible and therefore not reanalyzed in the HyperScale top-level run. This happens when there are wire couplings from the block interface paths to the internals of the block; the widening of the arrival window can result in different overlapping scenarios during the PrimeTime SI aggressor and victim analysis, result in additional crosstalk effects that are not fully accounted for during the block-level analysis of those internal victim paths.

Note: This HyperScale boundary constraint check for data_arrival is for the second order effects. The direct effect of a wider window reaching the block boundary is analyzed during top-level analysis because either the timing paths are already visible, or there are required times annotated at its fanout.

WHAT NEXT

A **data_arrival** violation does not require your action to fix the design or constraint; this is an **auto_fixable** violation. If the effects of wider windows need to be validated, save the updated and current context captured for the block of interest, and rerun HyperScale block-level analysis using this new context. The actual windows (wider than the previous block-level run) are applied and analyzed automatically.

EXAMPLES

The following verbose report shows violations in pin arrival times at block-level analysis that is different from the actual arrival times computed at the top level.

During block-level analysis, an expected min/max arrival window of 2.0 ns to 4.0 ns is defined for the analysis, but top-level analysis actual propagated arrival window is 1.8247 ns to 4.1472 ns, which is wider than the block-level setting. Therefore, you need to save this computed window as the actual context for the block and apply it to the block in the next HyperScale run.

```
pt_shell> report_constraint -boundary_check_include {data_arrival} \
          -all_violators -verbose

****************************************
Report : constraint
        -all_violators
        -verbose
        -boundary_check
Design : top
 ...
****************************************

HyperScale constraints report

   Constraint: data_arrival

   Instance  Pin(Port name)   Window type  CLK          Block   Top     Slack
   -------------------------------------------------------------------------
   core      core/u0/A(clk1)  max_rise     top_CLK1(r)  4.0000  4.1472  -0.1472
   core      core/u0/A(clk1)  min_rise     top_CLK1(r)  2.0000  1.8247  -0.1753
```

SEE ALSO

```
report_constraint(2)
auto_fixable_violations(3)
clock_latency_violations(3)
hier_enable_analysis(3)
input_slews_violations(3)
```

dbr_ignore_external_links

Specifies whether to ignore external links when reading in database files.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to its default of **false**, if the **read_db** command encounters an external link when reading a database (db) file, it extracts as much information as possible from the database (db) file so that the linker can restore the link. If you set this variable to **true**, the **read_db** command ignores external links and searches for an object by name only in the libraries set using the **link_path** variable.

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

SEE ALSO

```
link_design(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

This variable enables the use of a default operating condition per individual library. By default, this variable is set to **true**, and 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.

If you set this variable 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 set in the **link_path** variable).

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 area).

This variable is mainly for obtaining backward compatibility for the corner case of using default conditions in PrimeTime version T-2002.09 and earlier releases.

SEE ALSO

```
    link_path(3)
    set_operating_conditions(2)
```

delay_calc_waveform_analysis_constraint_arcs

Specifies whether setup and hold constraint analysis uses nonlinear delay model (NLDM) lookup tables instead of waveform propagation.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Set this variable to one of these values:

- **false** - If waveform propagation is enabled, setup and hold constraint analysis considers the waveform distortion effect. This analysis requires CCS timing and noise data in the library.
- **true** (true) - Setup and hold constraint analysis uses nonlinear delay model (NLDM) lookup tables and does not consider the waveform distortion effect.

SEE ALSO

```
extract_model(2)
report_timing(2)
update_timing(2)
delay_calc_waveform_analysis_mode(3)
```

delay_calc_waveform_analysis_mode

Controls usage of CCS-based waveform analysis for uncoupled and signal integrity calculation.

TYPE

string

DEFAULT

disabled

DESCRIPTION

You can set this variable to one of the following:

- **disabled** (default) - Disables CCS waveform analysis.
- **full_design** - Enables CCS waveform analysis on the entire design.

CCS waveform analysis requires libraries that contain CCS timing and CCS noise data. This feature, which includes uncoupled calculation, requires a PrimeTime SI license. However, you do not need to set the **si_enable_analysis** variable for uncoupled analysis.

SEE ALSO

```
report_timing(2)
update_timing(2)
si_enable_analysis(3)
```

design_attributes

Describes the predefined application attributes for design objects.

DESCRIPTION

analysis_type

Type: string
Returns the analysis type, either single or on_chip_variation.

area

Type: float
Returns the total area of the design. This is the sum of the areas of all leaf cells and nets.

capacitance_unit_in_farad

Type: float
Returns the unit of capacitance in the main library in farads. This attribute is read-only.

config_name

Type: string
Returns the name of the session saved for the current HyperScale run.

config_path

Type: string
Returns the HyperScale configuration path of the current HyperScale run.

context_config_name

Type: string
Returns the named session of HyperScale top context applied in HyperScale block-level runs.

context_config_path

Type: string
Returns the configuration path of HyperScale top context applied in HyperScale block-level runs.

context_timestamp

Type: string
Returns the timestamp of the HyperScale top context applied in HyperScale block-level runs.

current_unit_in_amp

Type: float
Returns the unit of capacitance in the main library in amps. This attribute is read-only.

designWare

Type: boolean
Returns true for a DesignWare design.

dont_touch

Type: boolean
Returns true if the design is excluded from changes. Values are undefined by default. Instantiations of designs with the dont_touch attribute set to true are not modified or replaced by ECO fixing commands (fix_eco_drc, fix_eco_power, and fix_eco_timing). This attribute is set by the set_dont_touch command and recognized by the ECO fixing commands (fix_eco_drc, fix_eco_power, and fix_eco_timing).

dynamic_power

Type: double
Returns the dynamic power of the design in watts. It is the sum of the dynamic power of all the cells of the design.

early_fall_cell_check_derate_factor

Type: float
Returns an early timing derating factor, specified by the set_timing_derate command, that applies to the design.

early_fall_clk_cell_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_fall_clk_net_delta_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_fall_clk_net_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_fall_data_cell_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_fall_data_net_delta_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_fall_data_net_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_rise_cell_check_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_rise_clk_cell_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_rise_clk_net_delta_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_rise_clk_net_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_rise_data_cell_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_rise_data_net_delta_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

early_rise_data_net_derate_factor

Type: float
Returns an early timing derating factor, specified using the set_timing_derate command, that applies to the design.

enable_bias

Type: boolean
Enables the UPF well bias mode when true. Set this attribute in the UPF command file with the following command:

```
set_design_attributes -elements {.} -attribute enable_bias true
```

extended_name

Type: string
Returns the complete, unambiguous name of the design. The extended_name of the design is the source_file_name attribute followed by a colon (:) followed by the full_name attribute. For example, the extended_name of design TOP read in from /u/user/simple.db is /u/user/simple.db:TOP.

full_name

Type: string
Returns the name of the design. For example, the full_name of design TOP read in from /abc/xyz/simple.db is TOP. This name can be ambiguous because several designs of the same name can be read in from different files.

gate_leakage_power

Type: double
Returns the gate leakage power of the design in watts. It is the sum of the gate leakage of all the cells of the design.

glitch_power

Type: double
Returns the glitch power of the design in watts. It is the sum of the glitch power of all the cells of the design.

internal_power

Type: double
Returns the internal power of the design in watts. It is the sum of the internal power of all the cells of the design.

internal_power_derate_factor

Type: float
Returns the internal power derating factor specified by the set_power_derate command that applies to the design.

intrinsic_leakage_power

Type: double

Returns the intrinsic leakage power of the design in watts. It is the sum of the intrinsic leakage of all the cells of the design.

is_context_available

Type: boolean
Returns true if HyperScale top context is available to be applied. This attribute is applicable only for HyperScale block runs. The attribute returns true after link and after update_timing only if top-level context exists for each configuration.

is_context_loaded

Type: boolean
Returns true if HyperScale top context has been loaded and applied during update_timing. This attribute is applicable only for HyperScale block runs. The attribute returns true after update_timing only if top-level context data was loaded during update_timing, even if the context was not fully applied because of override controls. The attribute is never true before update_timing.

is_current

Type: boolean
Returns true for the current design. This attribute changes for all designs when you use the current_design command.

is_edited

Type: boolean
Returns true if the design has been uniquified as a result of a netlist editing (ECO) change.

is_hyperscale_block

Type: boolean
Returns true if the current design is a HyperScale block.

is_hyperscale_top

Type: boolean
Returns true if the current design is a HyperScale top-level design. Mid-level HyperScale runs are considered to be both top and block.

late_fall_cell_check_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_fall_clk_cell_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_fall_clk_net_delta_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_fall_clk_net_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_fall_data_cell_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_fall_data_net_delta_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_fall_data_net_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_rise_cell_check_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_rise_clk_cell_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_rise_clk_net_delta_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_rise_clk_net_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_rise_data_cell_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_rise_data_net_delta_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

late_rise_data_net_derate_factor

Type: float
Returns a late timing derating factor, specified using the set_timing_derate command, that applies to the design.

leakage_power

Type: double
Returns the leakage power of the design in watts. It is the sum of the leakage power of all the cells of the design.

leakage_power_derate_factor

Type: float
Returns the leakage power derating factor specified by the set_power_derate command that applies to the design.

lower_domain_boundary

Type: boolean
Adds a lower-domain boundary in the boundary definition. You can set this attribute at any level of the hierarchy. To include lower-domain boundaries in the current scope, use the following command:

```
set_design_attributes -elements {.} -attribute lower_domain_boundary true
```

max_area

Type: float
Returns a floating-point number that represents the target area of the design. The units must be consistent with the units used from the logic library during optimization. The max_area value is set using the set_max_area command.

max_capacitance

Type: float
Returns the default maximum capacitance design rule limit for the design. The units must be consistent with those of the logic library used during optimization. It is set with the set_max_capacitance command.

max_fanout

Type: float

Returns the default maximum fanout design rule limit for the design. The units must be consistent with those of the logic library used during optimization. It is set with the set_max_fanout command.

max_transition

Type: float
Returns the default maximum transition design rule limit for the design. The units must be consistent with those of the logic library used during optimization. It is set with the set_max_transition command.

min_capacitance

Type: float
Returns the default minimum capacitance design rule limit for the design. The units must be consistent with those of the logic library used during optimization. It is set with the set_min_capacitance command.

min_fanout

Type: float
Returns the default minimum fanout design rule limit for the design. The units must be consistent with those of the logic library used during optimization.

min_transition

Type: float
Returns the default minimum transition design rule limit for the design. The units must be consistent with those of the logic library used during optimization.

object_class

Type: string
Returns the class of the object, which is "design". You cannot set this attribute.

operating_condition_max

Type: string
Returns the name of the maximum or single operating condition for the design. It is set with the set_operating_conditions command.

operating_condition_min

Type: string
Returns the name of the minimum operating condition for the design. This attribute is not valid in single operating condition analysis.

peak_power

Type: double
Returns the peak power of the design in watts. Note that the peak power of the design is not the sum of the peak power of the cells of the design because the cell peaks can occur at different times.

peak_power_end_time

Type: double
Returns the end time of the time interval in which peak power is measured for the design. The unit is nanoseconds (ns).

peak_power_start_time

Type: double
Returns the start time of the time interval in which peak power is measured for the design. The unit is nanoseconds (ns).

power_simulation_time

Type: float
Returns the total simulation time in averaged mode, in nanoseconds (ns).

power_states

Type: double
Returns the sum of the power_states of all the cells of the design.

process_max

Type: float
Returns the process value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. To associate operating conditions with the design, use the set_operating_conditions command.

process_min

Type: float
Returns the process value of the minimum operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. To associate operating conditions with the design, use the set_operating_conditions command.

rc_input_threshold_pct_fall

Type: float
Returns the characterization trip point (waveform measurement threshold) factor that the tool uses to calculate delays and transition times. Returns the value obtained from the main library (the first library in the link path).

rc_input_threshold_pct_rise

Type: float
Returns the characterization trip point (waveform measurement threshold) factor that the tool uses to calculate delays and transition times. Returns the value obtained from the main library (the first library in the link path).

rc_output_threshold_pct_fall

Type: float
Returns the characterization trip point (waveform measurement threshold) factor that the tool uses to calculate delays and transition times. Returns the value obtained from the main library (the first library

in the link path).

rc_output_threshold_pct_rise

Type: float
Returns the characterization trip point (waveform measurement threshold) factor that the tool uses to calculate delays and transition times. Specifies the value obtained from the main library (the first library in the link path).

rc_slew_derate_from_library

Type: float
Returns the slew derating factor that the tool uses to calculate delays and transition times. Specifies the value obtained from the main library (the first library in the link path).

rc_slew_lower_threshold_pct_fall

Type: float
Returns the characterization trip point (waveform measurement threshold) factor that the tool uses to calculate delays and transition times. Specifies the value obtained from the main library (the first library in the link path).

rc_slew_lower_threshold_pct_rise

Type: float
Returns the characterization trip point (waveform measurement threshold) factor that the tool uses to calculate delays and transition times. Specifies the value obtained from the main library (the first library in the link path).

rc_slew_upper_threshold_pct_fall

Type: float
Returns the characterization trip point (waveform measurement threshold) factor that the tool uses to calculate delays and transition times. Specifies the value obtained from the main library (the first library in the link path).

rc_slew_upper_threshold_pct_rise

Type: float
Returns the characterization trip point (waveform measurement threshold) factor that the tool uses to calculate delays and transition times. Specifies the value obtained from the main library (the first library in the link path).

resistance_unit_in_ohm

Type: float
Returns the unit of resistance in the main library in Ohms. This attribute is read-only.

source_file_name

Type: string
Returns the name of the file from which the design was read. For example, the source_file_name of design TOP read in from /abc/xyz/simple.db is /abc/xyz/simple.db.

switching_power

Type: double
Returns the switching power of the design in watts. It is the sum of the switching power of all the cells of the design.

switching_power_derate_factor

Type: float
Returns the switching power derating factor specified by the set_power_derate command that applies to the design.

temperature_max

Type: float
Returns the ambient temperature value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. To associate operating conditions with a design, use the set_operating_conditions command.

temperature_min

Type: float
Returns the ambient temperature value of the minimum operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. To associate operating conditions with a design, use the set_operating_conditions command.

time_unit_in_second

Type: float
Returns the unit of time in the main library in seconds. This attribute is read-only.

timestamp

Type: string
Returns the timestamp for the current HyperScale run. This attribute applies to top and block designs.

total_power

Type: double
Returns the total power of the design in watts. It is the sum of the total power of all the cells of the design.

tree_type_max

Type: string
Returns the tree_type value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. The tree_type value is used in prelayout interconnect delay estimation, and can have a value of best_case, balanced_case, balanced_resistance (cmos2 only), or worst_case.

tree_type_min

Type: string
Returns the tree_type value of the minimum operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. The tree_type value is used in prelayout interconnect delay estimation, and can have a value of best_case, balanced_case, balanced_resistance (cmos2 only), or worst_case.

violating_endpoints_max

Type: collection
Returns the timing path endpoints that have maximum delay violations, sorted in order of increasing slack.

violating_endpoints_min

Type: collection
Returns the timing path endpoints that have minimum delay violations, sorted in order of increasing slack.

voltage_max

Type: float
Returns the voltage value of the maximum or single operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. This attribute represents the supply voltage value for the operating condition.

voltage_min

Type: float
Returns the voltage value of the minimum operating condition for the design. The operating condition is defined in a library or by the create_operating_conditions command. You associate operating conditions with a design using the set_operating_conditions command. This attribute represents the supply voltage value for the operating condition.

voltage_unit_in_volt

Type: float
Returns the unit of voltage in the main library in volts. This attribute is read-only.

wire_load_min_block_size

Type: float
Returns the smallest hierarchical cell that has automatic wire load selection by area applied. If an automatic wire load selection group is specified as the default in the main library, or through the set_wire_load_selection_group command, it is applied to all hierarchical cells larger than the specified minimum block size.

wire_load_mode

Type: string
Returns the wire load model used to compute wire capacitance, resistance, and area for nets in a hierarchical design that has different wire load models at different hierarchical levels. Allowed values:

- top (default) - Uses the wire load model at the top hierarchical level.
- enclosed - Uses the wire load model on the smallest design that encloses a net completely.
- segmented - Breaks the net into segments, one within each hierarchical level. In the segmented mode, each net segment is estimated using the wire load model on the design that encloses that segment. The segmented mode is not supported for wire load models on clusters.

If no value is specified for this attribute, PrimeTime searches for a default in the first library in the link path. It is set with the set_wire_load_model command.

wire_load_model_max

Type: string
Returns the name of the design's wire load model for maximum conditions. It is set with set_wire_load_model.

wire_load_model_min

Type: string
Returns the name of the design's wire load model for minimum conditions. This attribute is not valid for single operating condition analysis. It is set with set_wire_load_model.

wire_load_selection_group_max

Type: string
Returns the name of the design's wire load selection group for maximum conditions. It is set with set_wire_load_selection_group.

wire_load_selection_group_min

Type: string
Returns the name of the design's wire load selection group for minimum conditions. It is set with set_wire_load_selection_group.

x_coordinate_max

Type: float
Returns the maximum x-coordinate of the area occupied by the design.

x_coordinate_min

Type: float
Returns the minimum x-coordinate of the area occupied by the design.

x_transition_power

Type: double
Returns the dynamic power used during transitions from the unknown (X) state to a known state (0 or 1).

y_coordinate_max

Type: float
Returns the maximum y-coordinate of the area occupied by the design.

y_coordinate_min

Type: float
Returns the minimum y-coordinate of the area occupied by the design.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

disable_case_analysis

Specifies whether case analysis is disabled.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, this variable is set to **false**, and constant propagation is performed in the design from pins either that are tied to a logic constant value or for those specified using the **set_case_analysis** command.

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 you set the **disable_case_analysis** variable to **true**, case analysis and constant propagation are not performed.

If you set the **disable_case_analysis_ti_hi_lo** variable to **true**, the tool does not propagate constants from pins that are tied to a logic constant value.

SEE ALSO

```
remove_case_analysis(2)
report_case_analysis(2)
set_case_analysis(2)
```

```
disable_case_analysis_ti_hi_lo(3)
```

disable_case_analysis_ti_hi_lo

Specifies whether logic constants are propagated from pins that are tied to a logic constant value.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, this variable is set **false**, and 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 you set this variable to **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_analysis** command.

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

SEE ALSO

```
remove_case_analysis(2)
report_case_analysis(2)
set_case_analysis(2)
```

```
disable_case_analysis(3)
```

distributed_cleanup_variables_for_swap

Controls cleaning up the state of variables when scenarios are swapped.

TYPE

string

DEFAULT

none

DESCRIPTION

If in Distributed Multi-Scenario Analysis (DMSA) fewer host processes are available than there are scenarios, the hosts run in scenario swapping mode in which they rotate and swap scenarios one after the other. PrimeTime's internal and user-defined variables set at one scenario executing at a particular host will not be removed or reset before loading of the image or the scriptware for another scenario when the scenario swap occurs.

To control the clean up of the state of PrimeTime's internal and user-defined variables during scenario image swapping, set the **distributed_cleanup_variables_for_swap** variable to one of these values:

none

- PrimeTime's internal and user-defined variables will not be automatically removed or reset during scenario image creation or swapping.

user_and_application

- PrimeTime will clean up the state of internal and user variables in the scenario swapping. The cleaning up process means that the variables at a DMSA worker process will be reset to their state recorded after sourcing of .synopsys_pt_setup files and executing of "pt_shell -x" arguments prior to loading up scenario image or scripts.

user

- PrimeTime will only remove or reset user variables (and will not reset internal variables) when creating a new scenario image prior to the scenario swapping.

application

- PrimeTime will reset up the state of internally defined variables only when creating a new scenario image prior to the scenario swapping.

This variable can only take effect if set in the DMSA Master process.

distributed_custom_protocol_error_detection_t

Specifies the maximum time that the distributed master waits for custom protocol worker processes to come online.

TYPE

integer

DEFAULT

86400

DESCRIPTION

This variable specifies the maximum time in seconds that the distributed master waits for custom protocol worker processes that were defined by **set_host_options -protocol custom**.

The default is 86400 seconds (24 hours).

SEE ALSO

`set_host_options(2)`

distributed_farm_protocol_error_detection_tim

Specifies the maximum time that the distributed master waits for a worker processes launched on a compute farm to come online.

TYPE

integer

DEFAULT

0

DESCRIPTION

This variable specifies the maximum time in seconds that the distributed master waits for worker processes that were defined by **set_host_options -protocol** with the **lsf**, **sge**, **rtda**, or **pbs** option.

The default of 0 means that no timeout is used.

SEE ALSO

`set_host_options(2)`

distributed_logging

Specifies the verbosity of logging during distributed analysis.

TYPE

string

DEFAULT

medium

DESCRIPTION

This variable controls the verbosity of the system logs written by the master and worker processes during distributed analysis.

After you start workers by using the **start_hosts** command, the master and worker processes write the following log files in the system_log directory in the distributed working directory:

Master log files

- /system_log/master.*hostname*.*processId*.*uniqueId*.bcast
  Contains the broadcast information the master shares with its workers in order for them to connect back to the master.
- /system_log/master.*hostname*.*processId*.err
  Contains the stderr output the master captures when launching worker processes, typically it is empty.
- /system_log/master.*hostname*.*processId*.log
  Contains logged information regarding master startup, worker process launching, task processing, and connection status.

Worker log file

- /system_log/worker.W*n*.*hostname*.*processId*.log
  Contains logged information regarding worker startup, task processing, and connection status.

In the names of the preceding log files:

- *hostname* - Name of the host on which the process is running
- *processId* - ID of the process
- *uniqueId* - Session specific ID
- *n* - Unique worker ID

To specify the verbosity of data written in the log files, set the **distributed_logging** variable to one of these values:

low

- The master log is compressed and contains minimal data relating to the master startup, worker launches, and task processing. You can access the compressed log content only after running the **stop_hosts** command or after the master PrimeTime session exits.
- No worker logs are generated.

medium

- The master log is uncompressed and contains minimal data relating to the master startup, worker launches, and task processing.
- No worker logs are generated.

high

- The master log is uncompressed and contains minimal data relating to the master startup, worker launches, task processing, and connection status.
- Each worker generates an uncompressed worker log that contains detailed worker startup, task processing, and connection status.

highest

- The master log is uncompressed and contains verbose data relating to the master startup, worker launches, task processing, and connection status.
- Each worker generates an uncompressed worker log that contains detailed worker startup, task processing, and connection status.

You must set the **distributed_logging** variable before launching workers. After you run the **start_hosts** command, you cannot change the value of the variable.

SEE ALSO

`start_hosts(2)`

distributed_sh_protocol_error_detection_timeo

Specifies the maximum time that the distributed master waits for sh, rsh, and ssh protocol worker processes to come online.

TYPE

integer

DEFAULT

300

DESCRIPTION

This variable specifies the maximum time in seconds that the distributed master waits for worker processes that were defined by **set_host_options -protocol** with the **sh**, **rsh**, or **ssh** option.

SEE ALSO

`set_host_options(2)`

eco_allow_filler_cells_as_open_sites

Specifies whether physically-aware ECO commands treat filler cells as open sites.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

The **eco_allow_filler_cells_as_open_sites** variable controls whether physically-aware ECO commands treat filler cells as open sites.

Filler cells are defined in the LEF (Library Exchange Format) file. In the LEF File, the CLASS construct specifies the type of macro used in the design. CORE macros are used to specify standard cells used in the core area of the design. A CORE macro can be one of the following types: FEEDTHRU, TIEHIGH, TIELOW, SPACER, and ANTENNACELL. Physically-aware ECO commands treat LEF CORE macros of type SPACER and FEEDTHRU as filler cells.

You can set this variable to one of these values:

- **true** (the default) - Physically-aware ECO treats all filler cells marked PLACED as open sites.
- **false** - Physically-aware ECO ignores filler cells.

You must set the **eco_allow_filler_cells_as_open_sites** variable before using the **set_eco_options** command. If you change the variable setting after using the **set_eco_options** command, the new setting has no effect on the physically-aware ECO behavior. If you are in a DMSA environment, you must set this variable in each worker using the **remote_execute** command.

SEE ALSO

set_eco_options(2)

eco_allow_insert_buffer_always_on_cells

Specifies whether always-on buffer insertion is allowed during ECO optimization.

TYPE

boolean

DEFAULT

false

DESCRIPTION

This variable specifies whether always-on buffers can be used during PrimeTime ECO optimization. It supports usage of buffers with the UPF-related library cell attribute **always_on** set to **true**. These buffers can be inserted by the **fix_eco_drc** or **fix_eco_timing** command dedicated to always-on buffer insertion.

Always on buffer-insertion is only performed when the net has at least one always_on cell at the driver or load. Buffer insertion does not check MV rules such as multi_voltage, upf, or power_domain.

By default, the variable is **false**, which causes ECO commands to skip always-on buffer insertion on all nets. When this variable is set to **true**, the **fix_eco_timing** and **fix_eco_drc** commands consider always-on buffering during optimization.

For distributed multi-scenario analysis (DMSA), set this variable in each slave.

For detailed information about the default behavior of UPF cells in ECOs, see the man pages of the **fix_eco_timing** and **fix_eco_drc** commands.

SEE ALSO

```
fix_eco_timing(2)
fix_eco_drc(2)
```

eco_allow_sizing_with_lib_cell_attributes

Specifies a list of library cell attributes that are enabled for sizing in ECO optimization.

TYPE

list

DEFAULT

"" (empty)

DESCRIPTION

This variable specifies a list of library cell attributes enabled for sizing in PrimeTime ECO optimization. Currently, it supports for the following four types of UPF related library cell attributes: **is_retention**, **is_level_shifter**, **always_on** and **is_isolation**. The attribute string matches in output of command **list_attributes** on library cell. By default, the variable is empty. PrimeTime ECO will skip those four type of cells for sizing. When this variable is set to sub list of the four library cell attributes, the **fix_eco_timing**, **fix_eco_drc**, and **fix_eco_power** commands will consider corresponding cells for sizing during optimization.

For distributed multi-scenario analysis (DMSA), set this variable in each slave.

For the detailed information about default behavior of UPF cells in ECO, see the man pages of the **fix_eco_timing**, **fix_eco_drc**, and **fix_eco_power** commands.

SEE ALSO

`fix_eco_timing(2)`

```
fix_eco_drc(2)
fix_eco_power(2)
list_attributes(2)
```

eco_alternative_area_ratio_threshold

Specifies the maximum allowable area increase when resizing a cell during the fixing process.

TYPE

float

DEFAULT

2.0

DESCRIPTION

This variable specifies the maximum allowable area increase when resizing a cell during the fixing process. By default, the cell size increase is limited to twice the area of the original cell. To override the default, set this variable to a positive value that specifies the maximum area increase as a ratio. If you want to resize without a maximum area limit, set the variable to 0.

Limiting the area increase can improve the timing predictability after physical implementation of an engineering change order (ECO), as greatly enlarged cells have a higher chance of perturbing the layout during incremental placement and routing. This can result in poor timing correlation after the implementation tool performs ECO changes.

SEE ALSO

```
fix_eco_drc(2)
fix_eco_timing(2)
```

eco_alternative_cell_attribute_restrictions

Specifies lib_cell attributes used to restrict cell sizing.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable restricts the choice of alternative library cells for the purposes of cell sizing. The variable accepts a whitespace delimited list of application or user-defined attributes on the lib_cell objects. A pair of library cells are considered equivalent if value of each of the attributes specified match. Note that the implicit cell equivalency checks performed by PrimeTime must also match. This variable affects all PrimeTime commands that perform cell sizing.

The following example of a usage model restricts cells to the same **area** (an application attribute) and the same **family** (a user-defined attribute):

```
pt_shell> define_user_attribute -classes lib_cell -type string family
pt_shell> set eco_alternative_cell_attribute_restrictions "area family"
```

SEE ALSO

```
define_user_attribute(2)
get_alternative_lib_cells(2)
report_alternative_lib_cells(2)
```

```
set_user_attribute(2)
size_cell(2)
```

eco_enable_fixing_clock_used_as_data

Enables buffer insertion at clock-used-as-data pins to fix hold timing violations.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, the **fix_eco_timing -type hold** command considers buffer insertion to fix hold timing violations at pins with the attribute **is_clock_used_as_data** set to **true**.

When this variable is set to **false** (the default), the **fix_eco_timing -type hold** command does not insert buffers at these pins.

For distributed multi-scenario analysis (DMSA), set this variable in the worker process, not the master.

SEE ALSO

`fix_eco_timing(2)`

eco_enable_mim

Enables ECO with multiply intantiated modules (MIMs).

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, the **fix_eco_timing**, **fix_eco_drc**, and **fix_eco_power** commands identify MIMs in the design and make same changes across the MIM instances.

If this variable is set to **true**, a PrimeTime-ADV license is checked out when fixing is performed.

For distributed multi-scenario analysis (DMSA), set this variable in the master.

For the detailed information about ECO with MIM, see the man pages of the **fix_eco_timing**, **fix_eco_drc**, and **fix_eco_power** commands.

SEE ALSO

```
fix_eco_timing(2)
fix_eco_drc(2)
fix_eco_power(2)
set_eco_options(2)
write_changes(2)
```

eco_enable_more_scenarios_than_hosts

Enables ECO fixing with more scenarios than available hosts.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **false** (the default), the **fix_eco_timing**, **fix_eco_drc**, and **fix_eco_leakage** commands require an equal number of scenarios and hosts.

To perform ECO fixing when the number of scenarios exceeds the number of available hosts, set the **eco_enable_more_scenarios_than_hosts** variable to **true**. However, the runtime and quality of results (QoR) might be affected because of limited resources.

SEE ALSO

```
fix_eco_drc(2)
fix_eco_leakage(2)
fix_eco_timing(2)
```

eco_estimation_output_columns

Specifies output columns to be displayed by the **estimate_eco** command for nonverbose reporting.

TYPE

string

DEFAULT

{area stage_delay arrival slack}

DESCRIPTION

This variable controls which output columns are displayed by the **estimate_eco** command for nonverbose reporting.

Considering an estimated stage that consists of the estimated cell, its driver pin, and its load pin, all on the slack critical path, the following keywords are available:

- **area** - Area of the specified library cell
- **stage_delay** - Delay from the input pin of the current cell to the most slack-critical load pin
- **arrival** - Arrival time at the most slack-critical load pin
- **slack** - Timing slack at the most slack-critical load pin
- **transition** - Transition time at the most slack-critical load pin
- **min_transition** - Worst min_transition design rule checking (DRC) for any pin in the estimated stage
- **min_capacitance** - Worst min_capacitance DRC for any pin in the estimated stage
- **max_transition** - Worst max_transition DRC for any pin in the estimated stage

- **max_capacitance** - Worst max_capacitance DRC for any pin in the estimated stage
- **max_fanout** - Worst max_fanout DRC for any pin in the estimated stage
- **data_pin_slack** - Timing slack at the data-pin of current sequential cell

The design rule constraint (DRC) keywords return the worst DRC value for all pins in the stage, including all load pins. For example, to see only area, slack, and maximum transition information, set the **eco_estimation_output_columns** variable to **area slack max_transition**.

SEE ALSO

`estimate_eco(2)`

eco_insert_buffer_search_distance_in_site_row

Specifies the maximum distance from a violating pin for open sites.

TYPE

integer

DEFAULT

8

DESCRIPTION

The **eco_insert_buffer_search_distance_in_site_rows** variable controls the maximum distance (specified in number of site rows) from a violating pin for open sites. Physically aware ECO commands search within this area for open sites when providing **insert_buffer** guidance.

A site row is defined in a physical block DEF file. Each physical block defines its own site row for cell placement. Each site row has a set width and height that is defined in LEF file for that block.

The actual distance searched around a violating pin is different in different physical blocks and is governed by the physical location of the violating pin, the physical block it lies in, and the site row height defined for that physical block.

By setting this variable, you can specify the search distance as a number of site rows without knowing the actual distance. In distributed multi-scenario analysis (DMSA), set the variable at the worker level, not the master level.

The default search area is an area of 8 site rows by 8 site rows around the violating pin.

Setting this variable to a very large number will adversely affect the runtime of physically aware ECO commands. The open site search area is automatically bound by the dimensions of the physical block in which the violating pin is located.

This variable controls the maximum search distance for **insert_buffer** guidance, but not **add_buffer_on_route** guidance.

SEE ALSO

```
fix_eco_drc(2)
fix_eco_timing(2)
```

eco_instance_name_prefix

Specifies the prefix used for naming new cell instances that are created by the **insert_buffer** command.

TYPE

string

DEFAULT

U

DESCRIPTION

By default, when creating new instances, the **insert_buffer** command uses the following instance names: U1, U2, U3, and so on.

To specify a different instance name prefix, set the **eco_instance_name_prefix** variable according to the following rules:

- The first character must be A-Z or a-z.
- Subsequent characters must be A-Z, a-z, 0-9, or an underscore (_).

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. Changing the value of this variable does not affect the current value of the instance number.

In the distributed multi-scenario analysis (DMSA) flow, specify the prefix name by setting this variable in the master. The tool uses the variable setting in the master regardless of the variable setting in the slaves.

EXAMPLES

Consider the case where the engineering change order (ECO) is made during the second iteration of timing closure. You can set this variable so that newly-created buffer and inverter instances are named in an easily-identifiable manner. For example,

```
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"}
```

SEE ALSO

insert_buffer(2)
eco_net_name_prefix(3)

eco_leakage_exclude_unconstrained_cells

Specifies whether to exclude unconstrained cells during leakage recovery.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the **fix_eco_leakage** command swaps unconstrained cells with preferred library cells during leakage recovery to maximize leakage power reduction.

To prevent the **fix_eco_leakage** command from using unconstrained cells during leakage recover, set the **eco_leakage_exclude_unconstrained_cells** variable to **true**.

In the distributed multi-scenario analysis flow:

- A cell is considered to be unconstrained only if it is unconstrained across all scenarios. If the cell is constrained in at least one scenario, the cell is considered to be constrained.
- You must set this variable in the master. The tool honors the variable setting only in the master and ignores the variable setting in the workers.

SEE ALSO

```
fix_eco_leakage(2)
```

eco_net_name_prefix

Specifies the prefix used for naming nets created by the **insert_buffer** command.

TYPE

string

DEFAULT

net

DESCRIPTION

By default, when creating new nets, the **insert_buffer** command uses the following net names: net1, net2, net3, and so on.

To specify a different net name prefix, set the **eco_net_name_prefix** variable according to the following rules:

- The first character must be A-Z or a-z.
- Subsequent characters must be A-Z, a-z, 0-9, or an underscore (_).

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. Changing the value of this variable does not affect the current value of the net number.

In the distributed multi-scenario analysis (DMSA) flow, specify the prefix name by setting this variable in the master. The tool uses the variable setting in the master regardless of the variable setting in the slaves.

EXAMPLES

Consider a case where the engineering change order (ECO) changes are being made during the second iteration of timing closure. You can set this variable so that newly-created buffer and inverter nets are named in an easily-identifiable manner. For example,

```
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"}
```

SEE ALSO

insert_buffer(2)
eco_instance_name_prefix(3)

eco_physical_match_site_row_names

Enables site-aware ECO which checks the site matching for cells and site rows.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, the **fix_eco_timing**, **fix_eco_drc**, and **fix_eco_power** commands identify site defined in LEF (Library Exchange Format) file. And the tool checks the site associated with the macro with the site rows defined in DEF (Design Exchange Format) file during physically aware ECO. User can use **set_eco_options** command to specify the site names.

For distributed multi-scenario analysis (DMSA), set this variable in the worker process, not the master.

For the detailed information about site-aware ECO, checks the man page of set_eco_options.

SEE ALSO

```
fix_eco_timing(2)
fix_eco_drc(2)
fix_eco_power(2)
set_eco_options(2)
```

eco_power_exclude_unconstrained_cells

Specifies whether to exclude unconstrained cells during power recovery.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the **fix_eco_power** command can resize or swap unconstrained cells with preferred library cells during power recovery to maximize power reduction.

To prevent the **fix_eco_power** command from using unconstrained cells during power recovery, set the **eco_power_exclude_unconstrained_cells** variable to **true**.

This variable has no effect on buffer removal by the **fix_eco_power** command. Only buffers that have both setup and hold constraints can be removed by the command.

In the distributed multi-scenario analysis (DMSA) flow:

- A cell is considered to be unconstrained only if it is unconstrained across all scenarios. If the cell is constrained in at least one scenario, the cell is considered to be constrained.
- You must set this variable in the master. The tool honors the variable setting only in the master and ignores the variable setting in the workers.

SEE ALSO

fix_eco_power(2)

eco_report_unfixed_reason_max_endpoints

Specifies the maximum number of violating endpoints listed in the unfixed reason report generated by the **fix_eco_timing** command.

TYPE

integer

DEFAULT

0

DESCRIPTION

This variable specifies the maximum number of violating endpoints listed in the unfixed reason report generated by the **fix_eco_timing** command. By default, the unfixed reason report is turned off. To enable the unfixed reason report, set this variable to a positive number. For the DMSA flow, you need to set this variable at the master instead of at the slave.

SEE ALSO

`fix_eco_timing(2)`

eco_save_session_data_type

Specifies violation types for ECO information to be saved and used by the **write_eco_design** command in Distributed Multi Scenario Analaysis (DMSA).

TYPE

string

DEFAULT

"" (empty)

DESCRIPTION

When this variable has a data type string, the **save_session** command saves additional ECO information, which the **write_eco_design** command in DMSA mode uses to create an ECO design.

One or more than one of the following keywords can be specified as valid string:

- **setup** - Setup timing violation
- **hold** - Hold timing violation
- **max_transition** - Max transition violation
- **max_capacitance** - Max capacitance violation
- **max_fanout** - Max fanout violation
- **noise** - Noise violation

The following keywords are also accepted as valid values:

- **timing** - Equals {setup hold}

- **drc** - Equals {max_transition max_capacitance max_faount}

The following example saves both setup and hold violation ECO information.

```
pt_shell> set eco_save_session_data_type "setup hold"
pt_shell> save_session my_session
```

The following example saves max_transition violation ECO information.

```
pt_shell> set eco_save_session_data_type max_transition
pt_shell> save_session my_session
```

The following example saves setup, hold, and noise violation ECO information.

```
pt_shell> set eco_save_session_data_type "setup hold noise"
pt_shell> save_session my_session
```

The following example saves setup, hold, and noise violation ECO information using the timing keyword.

```
pt_shell> set eco_save_session_data_type "timing noise"
pt_shell> save_session my_session
```

The following example saves setup, hold, max_transition, max_capacitance, max_fanout, and noise violation ECO information.

```
pt_shell> set eco_save_session_data_type "timing drc noise"
pt_shell> save_session my_session
```

SEE ALSO

write_eco_design(2)
read_eco_design(2)

eco_strict_pin_name_equivalence

Specifies whether only cells with identical pin names are equivalent for cell sizing.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), the tool can size cells even when the pin names of the cells are not identical. The tool uses other library cell information to match the pins of each cell.

In some cases, this behavior might not be desirable. If you set this variable to **true**, the tool considers cells to be equivalent only when the pin names on each cell match exactly.

SEE ALSO

```
get_alternative_lib_cells(2)
report_alternative_lib_cells(2)
size_cell(2)
```

eco_write_changes_prepend_libfile_to_libcell

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

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to its default of **false**, references to library cells in the **write_changes** command output contains only the library name and reference cell name information. For example,

```
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 you set this variable to **true**, the library's file name information is prepended to the library name to fully specify the library cell reference. For example,

```
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 multilibrary 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** command, which has already been executed and recorded.

Note that variable has no impact 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.

SEE ALSO

```
create_cell(2)
insert_buffer(2)
size_cell(2)
eco_write_changes_prepend_libname_to_libcell(3)
```

eco_write_changes_prepend_libname_to_libcell

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

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When you set this variable to **true**, references to library cells in the **write_changes** output contain library name and reference cell name information:

By default, the library's name is not prepended to the cell 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}
```

When this variable is set to **true**, the library's name is prepended to the cell name:

```
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}
```

Note that the variable controls whether the library name information is saved at the time of the engineering change order (ECO) operation. Changing the value of this variable does not modify the **create_cell**, **insert_buffer**, or **size_cell** command that has already been executed and recorded.

SEE ALSO

create_cell(2)
insert_buffer(2)
size_cell(2)
eco_write_changes_prepend_libfile_to_libcell(3)

enable_golden_upf

Enables the golden UPF flow.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When you set this variable to **true**, PrimeTime supports the golden UPF flow.

SEE ALSO

`load_upf(2)`

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 finishes processing a task, it returns the licenses it used to the master. When the **enable_license_auto_reduction** variable is set to its default of **false**, the master keeps the licenses checked out for future slave usage. When you set this variable to **true**, the master returns the licenses it 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 might not be able to acquire them again. This reduces the number of slaves that can execute concurrently leading to a degradation in performance during the subsequent analysis.

The **enable_license_auto_reduction** variable only has an impact if the master was launched with incremental license handling enabled. Incremental license handling can be enabled by setting the **SNPSLMD_ENABLE_INCREMENTAL_LICENSE_HANDLING** UNIX environmental variable to **1**.

To activate automatic license reduction, set the variable to **true**.

```
pt_shell> set_app_var enable_license_auto_reduction true
```

enable_path_tagging

Enables path tagging by the **create_path_tag_set** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, path tagging is disabled. To enable path tagging, set the **enable_path_tagging** variable to **true**.

SEE ALSO

```
create_path_tag_set(2)
get_timing_paths(2)
report_path_tag_set(2)
```

enable_rule_based_query

Enables or disables rule-based matching.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, rule-based matching is enabled. However, you must run the **set_query_rules** command first. If you set this variable to **true** without running the **set_query_rules** command first, the default query rules are used.

When this variable is set to **true**, the runtime for the query might be slower. Disable rule-based matching when it is no longer needed.

SEE ALSO

`set_query_rules(2)`

env_variables_violations

This man page describes **env_variables** hierarchical boundary check violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

An **env_variables** violation indicates differences in the environment variables between block- and top-level analysis. The tool checks only the application-defined Tcl variables that affect the results (QoR) of the timing analysis.

WHAT NEXT

This is a **non_auto_fixable** violation. You must resolve the difference in variable settings to ensure consistent results for the timing analysis.

EXAMPLES

The following verbose report shows a simple difference of top level enabling CCS calculation while block level uses non-CCS.

```
pt_shell> report_constraint -boundary_check_include {env_variables} -all_violators -verbose

****************************************
Report : constraint
         -verbose
Design : wrapper
 ...
****************************************

HyperScale constraints report

   Constraint: env_variables

   Instance      Attribute              Block     Top
   -------------------------------------------------------
   core2         rc_driver_model_mode   basic     advanced
```

SEE ALSO

```
report_constraint(2)
non_auto_fixable_violations(3)
```

extract_model_capacitance_limit

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

TYPE

float

DEFAULT

64

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, the **extract_model_capacitance_limit** variable is used to establish a more constraining design rule in the resulting timing model.

SEE ALSO

```
extract_model(2)
extract_model_clock_transition_limit(3)
extract_model_data_transition_limit(3)
extract_model_num_capacitance_points(3)
```

extract_model_clock_latency_arcs_include_all_

Determines whether all clock paths are included in the computation when creating clock tree latency or clock insertion delay arcs in the extracted timing models (ETM).

TYPE

string

DEFAULT

true

DESCRIPTION

If you set this variable to **false**, when extracting the clock tree latency, or clock insertion delay arcs, the PrimeTime model-generating commands traverse only the clock tree paths to the boundary registers. If this variable is set to **true** (the default), all clock tree paths, including those to the internal registers, are traversed.

This variable is effective only if the **extract_model_with_clock_latency_arc** variable is set to **true**.

The only modeling commands affected by this variable is the **extract_model** command. All formats of the extracted models are affected.

SEE ALSO

```
extract_model(2)
report_clock_timing(2)
extract_model_with_clock_latency_arcs(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 (in nanoseconds) 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.

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, this variable is used to establish a more constraining design rule in the resulting timing model.

SEE ALSO

```
extract_model(2)
extract_model_capacitance_limit(3)
```

```
extract_model_data_transition_limit(3)
extract_model_num_capacitance_points(3)
extract_model_num_clock_transition_points(3)
extract_model_num_data_transition_points(3)
```

extract_model_create_variation_tables

Decides whether the LVF tables are extracted into the ETM.

TYPE

string

DEFAULT

auto

DESCRIPTION

This variable controls what types of LVF tables will be created in the ETM when POCV is enabled.

When setting this variable to **auto** (the default), extract_model will generate delay sigma tables (ocv_sigma_cell_rise/fall). If slew variation is enabled, slew sigma tables will also be created. If variale timing_enable_constraint_variation is set to true, extract_model will also generate constraint sigma tables (ocv_sigma_rise/fall_constraint).

When setting this variable to **delay_only**, extract_model will only generate LVF tables for delay arcs but not for constraint arcs regardless whether constraint variation is enabled or not.

When setting this variable to **none**, extract_model will not generate any LVF tables. All the variation effects are embedded in the mean tables.

SEE ALSO

`extract_model(2)`

```
timing_enable_constraint_variation(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.

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.

SEE ALSO

extract_model(2)
extract_model_capacitance_limit(3)

```
extract_model_clock_transition_limit(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 in previous releases.

TYPE

string

DEFAULT

true

DESCRIPTION

ETMs generated by PrimeTime can be written in either .lib (Liberty) or .db (Synopsys database) format. Historically, the naming of the library in these two formats output can be different. The .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.

By default, this variable is set to **true**, and the preceding behavior is unchanged for backward compatibility.

If you set this variable to **false**, the .db format naming follows the .lib format. For example, both formats use the value specified for the **-output** option. This might improve scripting convenience when both the .lib and .db formats ETM models are mixed in the flow.

SEE ALSO

`extract_model(2)`

extract_model_enable_report_delay_calculatior

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

TYPE

string

DEFAULT

true

DESCRIPTION

If you set this variable 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.

This is enforceable only for the Synopsys database (.db) format output. For Liberty (.lib) format models, you can modify the files before compiling them to enable or disable the feature of the resulting library.

SEE ALSO

```
extract_model(2)
report_delay_calculation(2)
```

extract_model_gating_as_nochange

Controls the conversion from clock-gating setup and hold arcs into no-change arcs in the extracted model.

TYPE

string

DEFAULT

false

DESCRIPTION

When you set this variable to **true**, the clock gating setup and hold constraints are modeled as no-change arcs on the extracted model. When this variable is set to **false** (the default), clock gating checks are represented as separate setup and hold constraints.

This variable affects models in all output formats, such as the Synopsys database (.db) and Liberty (.lib) formats.

SEE ALSO

`extract_model(2)`
`extract_model_capacitance_limit(3)`

extract_model_include_clock_tree_pulse_width

Specifies whether the clock tree pulse width is included in the computation for minimum pulse width extraction or only the sequential clock pulse width is included.

TYPE

string

DEFAULT

false

DESCRIPTION

By default, the extract_model command includes only the sequential clock pulse width (the minimum pulse width on a clock pin) when extracting the minimum pulse width arcs.

To include the clock tree pulse width (the minimum pulse width on nonclock pins, typically for clock-gating cells), set the **extract_model_include_clock_tree_pulse_width** variable to **true**.

This variable affects only the **extract_model** command. All formats of the extracted models are affected.

SEE ALSO

```
extract_model(2)
report_min_pulse_width(2)
```

extract_model_include_ideal_clock_network_la

Controls the behavior of accounting user-defined network latencies for ideal clocks in the extracted timing models (ETM).

TYPE

string

DEFAULT

false

DESCRIPTION

By default, this variable is set **false**, and the extracted models treat ideal clock paths as zero delay in creating timing arcs, you are expected to reapply the same clock network latency values when the model is used.

If you set this variable to **true**, the PrimeTime model extraction uses the user-defined network latency for ideal clock paths leading to registers. This affects 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_edge and falling_edge delay arcs from ideal clock ports to output ports. It also affects the clock insertion delay arcs created in the model if extraction of such arcs are enabled by the **extract_model_with_clock_latency_arcs** variable.

This variable affects only the **extract_model** command and applies to all formats of the extracted models.

SEE ALSO

```
extract_model(2)
extract_model_with_clock_latency_arcs(3)
```

extract_model_include_upf_data

Controls whether the UPF data is extracted or merged in the models.

TYPE

string

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, the models generated by PrimeTime model extraction or merging contain UPF data reflecting the UPF data existing in the original design or models.

SEE ALSO

`extract_model(2)`
`merge_models(2)`

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 two 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 can 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, the extracted timing model (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 extracts them as regular setup and hold arcs.

If you set this variable to **true**, model extraction aligns 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.

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

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 Synopsys .db format models explicitly in the .lib format model files.

TYPE

string

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, the .lib format model files generated by PrimeTime model extraction has the same set of internal check pins that are created in the .db format models under certain conditions so that the PrimeTime timing engine correctly interprets the timing models.

By default, the variable is set to **false**. This might mean that the check pins need to be created by the down stream tools that interpret the .lib model files.

SEE ALSO

```
extract_model(2)
report_delay_calculation(2)
```

extract_model_merge_clock_gating

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

TYPE

string

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, the PrimeTime **extract_model** command merges clock gating setup and hold constraints with nonclock 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, ensure this variable is set to its default of **false**.

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, such as the I-V curve.

TYPE

float

DEFAULT

-1

DESCRIPTION

This variable controls the scale factor of the minimum index value used to create a steady-state current table (for example, I-V curve). This variable is a scale factor that is multiplied by VDD (the power supply voltage). For example, if VDD is 1.8, the minimum voltage used is -1.0*1.8 = -1.8.

SEE ALSO

```
extract_model(2)
report_noise(2)
extract_model_clock_transition_limit(3)
extract_model_data_transition_limit(3)
extract_model_noise_iv_index_upper_factor(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

Controls the scale factor of the minimum index value used to create a steady-state current table, such as the I-V curve.

TYPE

float

DEFAULT

2

DESCRIPTION

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

SEE ALSO

```
extract_model(2)
report_noise(2)
extract_model_clock_transition_limit(3)
extract_model_data_transition_limit(3)
extract_model_noise_iv_index_lower_factor(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_width_points

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

TYPE

string

DEFAULT

"" (empty string)

DESCRIPTION

Selects the exact noise width points of extracted noise immunity tables. The tool uses this variable only when you use the **extract_model -noise** command, and PrimeTime SI selects the library's noise immunity table for detecting noise on the input ports.

The value of this variable is a string of spaced floating values. For example,

```
set extract_model_noise_width_points "0.1 0.2 0.5 1.0"
```

SEE ALSO

extract_model(2)
report_noise(2)
extract_model_clock_transition_limit(3)
extract_model_data_transition_limit(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

integer

DEFAULT

5

DESCRIPTION

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

SEE ALSO

```
extract_model(2)
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_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

integer

DEFAULT

5

DESCRIPTION

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

SEE ALSO

```
extract_model(2)
extract_model_capacitance_limit(3)
extract_model_clock_transition_limit(3)
extract_model_data_transition_limit(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

integer

DEFAULT

5

DESCRIPTION

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

SEE ALSO

```
extract_model(2)
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, for example I-V curve, by defining the number of index, such as voltage, points in these tables.

TYPE

integer

DEFAULT

10

DESCRIPTION

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

The tool uses this variable only when you use the **extract_model -noise** command, and PrimeTime SI selects the library's steady-state current tables for detecting noise on the output and inout ports.

SEE ALSO

```
extract_model(2)
report_noise(2)
extract_model_clock_transition_limit(3)
extract_model_data_transition_limit(3)
extract_model_noise_iv_index_lower_factor(3)
extract_model_noise_iv_index_upper_factor(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_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 tool uses this variable only when you use the **extract_model -noise** command, and PrimeTime SI selects the library's noise immunity tables for detecting noise on the input ports.

SEE ALSO

```
extract_model(2)
report_noise(2)
extract_model_clock_transition_limit(3)
extract_model_data_transition_limit(3)
extract_model_noise_width_points(3)
extract_model_num_clock_transition_points(3)
extract_model_num_data_transition_points(3)
```

extract_model_short_syntax_compatibility

Controls whether the short groups are generated in extracted timing model for those feedthrough wires.

TYPE

string

DEFAULT

true

DESCRIPTION

When this variable is set to true, extract_model will write short syntax for the feedthrough wires. When extract_model_upf_supply_precedence is set to netlist, extract_model will not assign related_power/ground_pin or related_bias_pin for the shorted ports. The timing arcs are not affected by this variable.

This variable affects models in all output formats, such as the Synopsys database (.db) and Liberty (.lib) formats.

SEE ALSO

```
extract_model(2)
extract_model_upf_supply_precedence(3)
```

extract_model_single_pin_cap

Provides backward compatibility with the introduction of the minimum, maximum, rise, and fall specific pin capacitances to account for Miller effect.

TYPE

string

DEFAULT

true

DESCRIPTION

This variable is needed to deal with libraries accounting for Miller Effect by having different minimum, maximum, rise, and fall capacitance values for pins of library cells. The actual timing arcs extracted is not affected by this variable and still accounts for the Miller effect if it is available in the library and applicable in the analysis condition. For minimum and maximum paths and rise and 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.

You can set this variable to one of these values:

- **true** - The models extracted keep only a single capacitance value for a pin. The capacitance written is the maximum of all the minimum, maximum, rise, and fall values.
- **false** - If you are using the .lib and .db file formats, different minimum, maximum, rise, and fall pin capacitance values are extracted, if available and applicable. These values are written as per the .lib and .db syntax.

This variable affects only the **extract_model** command.

SEE ALSO

extract_model(2)

extract_model_single_pin_cap_max

Provides a method to write only minimum capacitance values in a model that is in single capacitance mode.

TYPE

string

DEFAULT

true

DESCRIPTION

This variable is needed to deal with libraries accounting for Miller Effect by having different minimum, maximum, rise, and fall capacitances for pins of library cells. The actual timing arcs extracted is not affected by this variable and still accounts for the Miller effect, if available in the library and applicable in the analysis condition. For minimum and maximum paths and rise and 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 a "single" operating condition.

The **extract_model_single_pin_cap_max** variable takes effect only if the **extract_model_single_pin_cap** variable is set to **true**. You can set **extract_model_single_pin_cap_max** to one of these values:

- **true** - Writes the maximum capacitance to the model.
- **false** - Writes the minimum capacitance to the model.

This variable affects only the **extract_model** command.

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 and hold arcs in the extracted timing model (ETM).

TYPE

string

DEFAULT

false

DESCRIPTION

You can set this variable to one of these values:

- **false** - Merges all clock gating checks and attaches them to the same pin.
- **true** - Detects clock gating setup and hold constraints that cannot be paired together; creates a separate internal check pin to avoid difference in timing analysis with the model in PrimeTime.

This variable affects models in all output formats, such as the Synopsys database (.db) and Liberty (.lib) formats.

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 the Library to define such arc pairing. Therefore, the pairing must occur automatically by PrimeTime when library 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 opposite edge

has been prechosen as the most constraining or 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 constraint in choosing the most constraining 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 endpoints that contribute to the constraints between them. ETM must evaluate the 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, the details of the original arc and path pairing relationship is lost, resulting in potentially different arc pairing when the ETM is used versus during netlist timing. Consequently, model validation can show a mismatch due to different clock edges are used for the originally mispairing check. The difference arises most commonly when:

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

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

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

string

DEFAULT

none

DESCRIPTION

This variable controls the number of progress messages shown during the timing model extraction process.

You can set this variable to one of these values:

- **none** - Does not show progress messages.
- **low** - Shows messages at the beginning of major phases of the **extract_model** command.
- **medium** - Shows all messages for **low** and messages at the beginning of extraction subphases that might incur significant processing time.
- **high** - Shows all messages for **medium** and percentage complete messages for long running subphases.

SEE ALSO

`extract_model(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 you set this variable to **true**, the generated models do not contain the **is_three_state** pin attribute.

The **extract_model** command adds the **is_three_state** pin attribute to the .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. The tool assumes that only one of the three state drivers is driving at a time.

SEE ALSO

`extract_model(2)`

extract_model_upf_supply_precedence

Controls the precedence of **set_port_attributes**, **set_related_supply_net**, and netlist logic in determining the **related_power_pin** and **related_ground_pin** attributes of a pin in an ETM.

TYPE

string

DEFAULT

netlist

DESCRIPTION

This variable controls how the values are determined for the **related_power_pin** and **related_ground_pin** attributes.

Setting this variable to **netlist** (the default) specifies the following precedence, from highest to lowest:

1. Netlist logic
2. **set_port_attributes -receiver_supply** for input and **set_port_attributes -driver_supply** for output
3. Primary supply net of the power domain

Setting this variable to **internal** specifies the following precedence, from highest to lowest:

1. **set_port_attributes -receiver_supply** for input and **set_port_attribute -driver_supply** for output
2. Netlist logic
3. Primary supply net of the power domain

Setting this variable to **external** specifies the following precedence, from highest to lowest:

1. **set_port_attributes -driver_supply** for input and **set_port_attributes -receiver_supply** for output
2. **set_related_supply_net**
3. Netlist logic
4. Primary supply net of the power domain

SEE ALSO

```
extract_model(2)
set_port_attributes(2)
set_related_supply_net(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 the worst-slew propagation mode.

TYPE

string

DEFAULT

false

DESCRIPTION

When this variable is set to **false**, no adjustment is made in the extraction, and the models created are the same as before extraction. When you set the variable to **true**, the extracted tables are adjusted with the current context slew, if it is more conservative to do so. This results in a model that generally passes the model validation better if the netlist is also timed in the worst-slew propagation mode. The adjusted model is generally more pessimistic compare to the model extracted when the variable is set to **false**.

The only modeling command affected by this variable is the **extract_model** command. All formats of the extraced models are affected.

SEE ALSO

`extract_model(2)`

extract_model_with_3d_arcs

Enables or disables creating models contain arcs with three-dimensional delay and transition tables.

TYPE

string

DEFAULT

true

DESCRIPTION

When this variable is set to its default of **true**, the PrimeTime model-generating commands attempt to merge certain output-to-output delays, clock-to-output, or both arcs into three-dimensional arcs with related_output_load as the third variable in the delay table, transition table, or both tables.

When you set this variable to **false**, the model keeps all output-to-output, clock-to-output, or both constraint arcs as they are, which is the default behavior of model extraction for PrimeTime version T-2002.09 and earlier releases.

The only modeling command affected by this variable is the **extract_model** command. All formats of the extracted models are affected.

If your downstream tools do not know how to handle timing arcs with three-dimensional delay tables, set this variable to **false** before extracting the timing model.

SEE ALSO

```
extract_model(2)
```

extract_model_with_ccs_timing

Controls whether CCS timing data is generated in extracted timing models.

TYPE

string

DEFAULT

false

DESCRIPTION

This variable controls whether CCS timing data is generated in an extracted timing model (ETM).

When this variable set to **false** (the default), the **extract_model** command extracts only NLDM tables for delay arcs. When this variable is set to **true**, the **extract_model** command also extracts CCS receiver and driver model tables, in addition to NLDM delay tables.

CCS timing and noise modeling data are needed in an ETM to support advanced waveform propagation analysis at the higher level of hierarchy where the model is used. In that case, set the variable to **true**. Otherwise, leave it set to false to skip CCS modeling and extract only NLDM delay modeling, which reduces the extraction runtime and produces a smaller .lib file.

This variable affects models in all output formats, such as the Synopsys database (.db) and Liberty (.lib) formats.

SEE ALSO

```
    extract_model(2)
    delay_calc_waveform_analysis_mode(3)
```

extract_model_with_clock_latency_arcs

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

TYPE

string

DEFAULT

false

DESCRIPTION

When you set this variable to **true**, the PrimeTime model-generating commands 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.

By default, this variable is set to **false**, and the extracted models do not have clock insertion delay arcs in them.

This variable affects only the **extract_model** command; it applies to all formats of the extracted models.

If you extract .lib format models with clock latency arcs, you must compile the model with Library Compiler version V-2003.12 or later.

The clock latency arcs compensate and balance clock tree skews at chip level. Those clock latency arcs in the models are also considered by the **report_clock_timing** PrimeTime command when reporting the clock tree latency and skews. Commands such as **report_delay_calculation**, **set_timing_derate**, **set_annotated_delay**, and **get_timing_arcs** also recognize the new insertion delay arcs.

SEE ALSO

```
extract_model(2)
report_clock_timing(2)
extract_model_clock_latency_arcs_include_all_registers(3)
```

extract_model_with_min_max_delay_constrain

Enables or disables creating models contain set_min/max_delay constraints.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

You can set this variable to one of these values:

- **false** - PrimeTime model-generating commands ignore all **set_min_delay** and **set_max_delay** constraints; you must delete these constraints before extracting the model to pass the model validation.
- **true** - PrimeTime model-generating commands create internal check pins and construct arcs between ports and internal check pins. PrimeTime also writes the **set_min_delay** and **set_max_delay** constraints into constraint files.

This variable affects only the **extract_model** command and all formats of the extracted models.

SEE ALSO

```
extract_model(2)
set_max_delay(2)
set_min_delay(2)
```

extract_model_write_case_values_to_constrair

Controls whether to write logic values separately into the extracted timing model (ETM) constraint file due to the user-defined case analysis value propagation.

TYPE

string

DEFAULT

false

DESCRIPTION

If this variable is set to its default of **false**, all logic constant values reaching block I/O ports are written into the .lib and .db files as a function attribute for the pin. This occurs regardless of whether the logic value is due to propagate circuit intrinsic functional constants or user-defined case analysis values. This is the behavior of the ETM.

If you set this variable 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-defined analysis settings, the logic value is written into the ETM constraint file with the proper **set_case_analysis** command.

This variable affects models in all output formats, such as Synopsys database (.db) and Liberty (.lib) formats.

An ETM generated by PrimeTime, captures I/O timing behavior for the purpose of static timing analysis. 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. By design, the ETM does not retain any functional information about the original netlist and is essentially a functional black-box. Lacking functional definition, an ETM by itself might not be well suited for design steps where the cell function is of primary concern. However, because logic values and their propagation impact timing of both the block and higher level netlist where the model becomes instances, ETMs have 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-defined case analysis values. PrimeTime represents the logic values of the macro block with the simple function attribute on pins. This ensures consistency from a timing analysis perspective. For certain design flows where some tools consuming ETMs in certain steps need to be able to differentiate logic values resulted from functional constant versus case analysis propagation. Optionally, it is possible to write logic values resulting from case analysis to a separate constraint file, and write only the functional constant with function attribute.

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

SEE ALSO

```
extract_model(2)
set_case_analysis(2)
```

extract_model_write_extended_moments

Controls whether to write extended moments in ETM Lib file, when data is available.

TYPE

boolean

DEFAULT

true

DESCRIPTION

This variable controls whether to write extended moments in ETM Lib file, when data is available. This variable is only valid and effective when the **extract_model_create_variation_tables** is set to **auto** or **delay_only**.

When this variable is set to **true** (the default), the **extract_model** command writes extended moments, if 4-value POCV calculations are enabled. The .lib will contain the new moments LVF syntax for all variation types enabled. When this variable is set to **false**, the **extract_model** command will only write 2-value early/late LVF tables. If calculations are using 4-value, the output will be converted into "bounding" mean/sigma LVF tables.

SEE ALSO

```
extract_model(2)
extract_model_create_variation_tables(3)
```

extract_model_write_verilog_format_wrapper

Writes Verilog format wrapper for wrapper plus core style ETM.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

You can set this variable to one of these values:

- **false** - The **extract_model** command creates a database format wrapper for wrapper plus core style extracted timing model (ETM).
- **true** - The **extract_model** command creates Verilog format wrapper is created instead.

SEE ALSO

`extract_model(2)`

filter_collection_extended_syntax

TYPE

boolean

DEFAULT

application specific

DESCRIPTION

This variable controls whether the filter_collection command supports extended math expressions. Please see the man page for filter_collection for details.

SEE ALSO

filter_collection(2)

gca_setup_file

Specifies the name of the file that is read by the In-Design PrimeTime GCA flow before constraint analysis.

TYPE

string

DEFAULT

"" (empty)

DESCRIPTION

This variable specifies the name of a Tcl setup file to be read by the In-Design PrimeTime GCA flow before constraint analysis. You can specify user-defined constraint checking rules and violation waivers for PrimeTime GCA in this setup file. PrimeTime GCA sources this file after loading the design and constraints.

Note:

This setup file is read by PrimeTime GCA at a different time than the .synopsys_gca.setup initialization file, which is loaded during the startup of PrimeTime GCA.

By default, this variable is set to an empty string, and no setup file is read after design loading during the In-Design PrimeTime GCA flow.

SEE ALSO

`check_constraints(2)`

global_timing_derate_violations

This man page describes **global_timing_derate** hierarchical boundary check violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **global_timing_derate** violation indicates differences in the timing derating applied between block- and top-level analysis.

During block-level analysis, the global timing derating values are captured and saved, this includes both design-level timing derating values and library cell derating values, but does not include instance specific derating settings. These saved values are used to check against top-level analysis settings applicable to the block instances.

Exact matching is not required. If block-level derating results in more conservative timing analysis, the tools reports no violations. Instead, the tools reports a violation if top-level analysis requires more margin than block-level analysis.

WHAT NEXT

This is a **non_auto_fixable** violation. To resolve the differences in timing deratings, either apply the same derating as top-level for block-level analysis, or use more conservative settings at the block level than at the top level.

Also note that violations reported for library cell derating values can also arise from the libraries being different. For more information about library mismatches between block- and top-level analysis, use the **library_mapping** command. For those cases, aligning the libraries might automatically resolve these library cell-related timing derating mismatches.

EXAMPLES

The following verbose report shows global design timing deratings and library-cell specific deratings (FD2 in the lsi_10k library). Deratings applied at block-level analysis show smaller margins than those applied at top-level analysis.

```
pt_shell> report_constraint -boundary_check_include {global_timing_derate} \
          -all_violators -verbose

****************************************
Report : constraint
        -all_violators
        -verbose
        -boundary_check
Design : top
 ...
****************************************

HyperScale constraints report

   Constraint: global_timing_derate

   Instance      Derate type            Window type    Block    Top     Slack
   ---------------------------------------------------------------------------
   md            static_data_net        min_rise       0.9      0.8     -0.1
   md            static_data_net        min_fall       0.9      0.8     -0.1
   md            static_clock_net       min_rise       0.9      0.8     -0.1
   md            static_clock_net       min_fall       0.9      0.8     -0.1
   md            dynamic_data_net       max_rise       1        1.1     -0.1
   md            dynamic_data_net       max_fall       1        1.1     -0.1
   md            dynamic_data_net       min_rise       0.95     0.9     -0.05
   md            dynamic_data_net       min_fall       0.95     0.9     -0.05
   md            dynamic_clock_net      max_rise       1        1.1     -0.1
   md            dynamic_clock_net      max_fall       1        1.1     -0.1
   md            dynamic_clock_net      min_rise       0.95     0.9     -0.05
   md            dynamic_clock_net      min_fall       0.95     0.9     -0.05
   md            cell_check             max_rise       1.34     1.38    -0.04
   md            cell_check             max_fall       1.34     1.38    -0.04
   lsi_10k:FD2   library_cell_data      min_rise       0.81     0.79    -0.02
   lsi_10k:FD2   library_cell_data      min_fall       0.81     0.79    -0.02
   lsi_10k:FD2   library_cell_clock     min_rise       0.81     0.79    -0.02
   lsi_10k:FD2   library_cell_clock     min_fall       0.81     0.79    -0.02
   lsi_10k:FD2   library_cell_check     max_rise       1.44     1.46    -0.02
   lsi_10k:FD2   library_cell_check     max_fall       1.44     1.46    -0.02
```

SEE ALSO

report_constraint(2)
report_timing_derate(2)
set_timing_derate(2)
non_auto_fixable_violations(3)

golden_upf_report_missing_objects

Specifies whether to report missing object errors in the golden UPF flow.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

All UPF tools are expected to suppress missing object errors during golden UPF reapplication with a few exceptions. If you set this variable to **true**, PrimeTime displays information messages for missing objects.

SEE ALSO

```
load_upf(2)
enable_golden_upf(3)
```

gui_object_attributes

Describes the predefined application attributes for gui_object objects.

DESCRIPTION

full_name

Type: string
Returns the full name of the object.

name

Type: string
Returns the name of the object.

object_class

Type: string
Returns the class of the object, which is a constant equal to gui_object. You cannot set this attribute.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

hier_characterize_context_mode

Sets the context characterization mode to perform either full context characterization or only extraction of constraints without timing analysis.

TYPE

string

DEFAULT

full_context

DESCRIPTION

This variable specifies the mode of characterization driven by the **characterize_context** and **write_context** commands. The variable setting also affects the behavior of other commands such as **update_timing** and **read_parasitics**.

The variable can be set to either **full_context** or **constraints_only**.

full_context (the default)

The **full_context** mode is the full-accuracy context characterization mode. In this mode, there are no impacts to regular PrimeTime data reading and timing analysis. Use the **characterize_context** command to select the block instances for which the tool performs additional steps to capture or preserve the full timing context computed during timing analysis. You can write out the context data with the **write_context** in several formats.

constraints_only

This **contraints_only** setting selects the constraint extraction mode in which the tool perform minimal analysis during **update_timing** and automatically skips unnecessary data loading steps (parasitics, AOCV/POCV tables, annotated delays, and so on) and disables unnecessary features (SI, POCV, CRPR,

and so on). Use the **characterize_context** command to specify the block instances from which to extract a set of full block-level constraints. Then use the **write_context** command to write out the constraints in script format, either constraints.pt and variables.pt or constraints.sdc and variables.sdc.

The **constraints_only** mode limits the **update_timing** command to only the steps needed to extract the block constraints, such as the propagation of clocks, constants, and timing exceptions. This mode is recommended only for writing out block constraints, and not for any detailed timing analysis or optimization.

This mode is incompatible with HyperScale analysis and cannot be used in any session in which HyperScale analysis is performed.

Use a separate PrimeTime session for **constraints_only** constraint extraction, as in the following script:

```
set_app_var hier_characterize_context_mode constraints_only
read_verilog Counter.v
read_verilog Top.v
link_design
 ...
characterize_context -block Counter
 ...
update_timing   # Constraint extraction only,
 ...            # no timing information generated
write_context -format ptsh Counter -output $cnsDir
```

Note: This variable affects the behavior of the **link_design** command, so you need to set it before linking.

SEE ALSO

```
characterize_context(2)
link_design(2)
write_context(2)
```

hier_clock_mapping_period_percent_tolerance

Specifies a tolerance for automatic clock mapping that allows clocks with different frequencies or duty cycles to be mapped as equivalent in HyperScale hierarchical analysis.

TYPE

float

DEFAULT

0

DESCRIPTION

By default, HyperScale hierarchical analysis automatically maps top-level and block-level clocks as equivalent when the clock characteristics match exactly. To allow automatic mapping between clocks that have slightly different periods or duty cycles, set this variable to the desired tolerance.

Specify the tolerance as a decimal fraction of the clock period and duty cycle. For example, to allow a difference of 2 percent, enter 0.02 as the tolerance value. In that case, the smaller period must be within 2 percent of the larger period, and the smaller duty cycle must be within 2 percent of the larger duty cycle, for the clocks to be considered equivalent for mapping purposes.

If all other conditions for mapping clocks are met, and the period and duty cycle are within the threshold difference, the clocks are automatically mapped.

SEE ALSO

`report_clock(2)`

```
set_clock_map(2)
set_hier_config(2)
hier_disable_auto_clock_mapping(3)
timing_save_hier_context_data(3)
```

hier_constraint_write_context

Specifies whether the HyperScale top-level flow writes the ASCII boundary I/O context.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether the tool writes ASCII I/O context in the HyperScale top-level flow. By default, the tool writes out binary context. If you set this variable to **true**, the HyperScale top-level flow also writes out the ASCII I/O context.

SEE ALSO

```
save_session(2)
set_hier_config(2)
hier_enable_analysis(3)
```

hier_context_merge_strict_clock_equivalence

Controls HyperScale context merging behavior for MIM instances that are not fully mergeable.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, HyperScale top-level analysis enforces exact matching of clocks across all multiply instantiated modules (MIMs) being merged for hierarchical analysis and constraint extraction. If there is any mismatch for a clock on a certain instance, that instance is completely excluded from context merging for all types of context data.

The **hier_context_merge_strict_clock_equivalence** variable lets you modify this behavior. It is set to **true** by default. Set the variable to **false** to relax the strict requirements and allow a module with mismatching context to be merged in certain cases.

The PrimeTime tool automatically performs context data merging during HyperScale analysis when you specify more than one instance of a given block with the same configuration. By default, constraint and context data merging can be performed only when the clocks are exactly matched across the instances.

To decide whether two instances are mergeable, all boundary and internal real clocks are checked and matched between the instances. Two instances are considered unmergeable if no match can be found between one or more clocks. This checking is done across all MIM instances, and the mergeable instances are clustered as a group. By default, if no MIM reference is specified (the **-mim_reference** option is not used in the **set_hier_config** command), the cluster with the largest number of instances is used for context merging, and the instances not in this cluster are excluded from merging.

When you set this variable to **false**, instead of completely dropping the remaining instances outside of the

cluster unconditionally, the tool considers merging them. If the context difference is not clock-related, the block is merged. If the context difference is clock-related, the block is merged if a matching clock can be found from the reference instance set; or if no matching clock can be found, the block is still considered unmergeable and the relevant context from the instance is dropped.

SEE ALSO

```
save_session(2)
set_hier_config(2)
update_timing(2)
hier_merge_match_clock_with_constant(3)
```

hier_create_template_scripts

Enables flat to hyperscale scripts generation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When you set this variable to **true**, flat to hyperscale scripts generation flow is enabled within PrimeTime. In this flow, PrimeTime reads in the flat script and generates hyperscale run scripts. Besides this, this flow also performs constraint extraction so that user can directly run the hyperscale analysis flow without doing constraint extraction. Note users need to call **write_hier_data** to trigger the flow. Hyperscale scripts and constraints will be generated in the directory specified by **write_hier_data**.

Enabling **hier_create_template_scripts** affects the **link_design**, **update_timing** and **save_session** commands.

Note: Because the **link_design** command is controlled by this variable, you must set the **hier_create_template_scripts** variable before linking. After the design is linked, further changes to this variable results in error, and it has no effect.

SEE ALSO

`link_design(2)`

```
save_session(2)
set_hier_config(2)
update_timing(2)
write_hier_data(2)
```

hier_data_version

Indicates the PrimeTime HyperScale database version that the application currently runs.

TYPE

string

DESCRIPTION

This read-only variable is set to the HyperScale database version of the application currently running.

SEE ALSO

`save_session(2)`

hier_disable_auto_clock_mapping

Disables automatic clock mapping in the HyperScale hierarchical analysis flow.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, HyperScale hierarchical analysis performs automatic mapping of clocks between the top level and block level by looking for matching clock characteristics.

To disable automatic clock mapping, set the **hier_disable_auto_clock_mapping** variable to **true**. This is useful when you want to perform interactive or fully manual clock mapping. With this setting, you need to map clocks by using the **set_clock_map** command. If you have a HyperScale constraint extractor run, you can do this by sourcing the clock_map.pt file. Unspecified clock mappings remain unmapped.

SEE ALSO

```
report_clock(2)
set_clock_map(2)
set_hier_config(2)
hier_clock_mapping_period_percent_tolerance(3)
timing_save_hier_context_data(3)
```

hier_distributed_working_directory

Specifies the working directory for HyperScale distributed analysis. The directory must be accessible to the distributed farm hosts for read and write operations during timing analysis.

TYPE

string

DEFAULT

""

DESCRIPTION

You need to set this variable to perform distributed HyperScale analysis or use the flat reporting flow. The specified directory is used to store the intermediate runtime data as well as logs and reports.

The specified directory must be network accessible for all the distributed farm hosts launched with the **set_host_options** command. The **hier_distributed_working_directory** variable must be set before you use the **start_hosts** command.

SEE ALSO

```
save_session(2)
set_hier_config(2)
set_host_options(2)
start_hosts(2)
update_timing(2)
hier_enable_analysis(3)
```

hier_enable_analysis

Enables the HyperScale hierarchical analysis flow.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable the HyperScale hierarchical analysis flow in the PrimeTime tool.

In the HyperScale flow, the tool analyzes the block-level and top-level portions of the design using separate runs and accurately handles the timing interfaces across hierarchical boundaries, significantly improving the capacity and performance of hierarchical timing analysis without impacting the quality of results.

Enabling HyperScale analysis affects the behavior of the **link_design**, **update_timing**, **save_session**, and **report_constraints** commands.

Because the behavior of the **link_design** command depends on whether the HyperScale flow is enabled, you must set the **hier_enable_analysis** variable before you link the design.

When HyperScale analysis is enabled, use the **set_hier_config** command to specify the HyperScale analysis parameters.

SEE ALSO

```
link_design(2)
report_constraint(2)
save_session(2)
set_dont_override(2)
set_hier_config(2)
update_timing(2)
```

hier_enable_detailed_side_input_path_timing

Specifies if PrimeTime HyperScale analysis should capture the full details of paths starting from side inputs (internal startpoint).

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Side input path is a HyperScale specific behavior. In the HyperScale analysis flow, there are block-level and top-level analyses. Side input paths at block level are internal register to register paths, and they converge with interface paths by sharing some common logic through some multiple-input cells, For example, an AND gate whose A pin is on an interface path starting from a block port and leading to some register, while its B pin is on a path starting from a register cell. During HyperScale block data reduction, paths through the A pin are retained while the fanin of B pin is removed. This pin is the side input pin of the interface path. To have the same accuracy of flat analysis at the HyperScale top level for the paths leading to the common register of the AND gate, the tool captures and annotates all the timing data propagated at B pin in binary format. At the top level, the paths appear to be starting from these dangling side input pins.

When this variable is set to **false** (the default), the paths starting from these side inputs maintain their effects to the interface paths, such as worst slew and SI crosstalk windows, but themselves are false paths at top level.

When this variable is set to **true**, detailed arrival times are captured in the model to reproduce the slacks of block internal timing paths end at interface registers. This detailed model has runtime and memory impact to both block and top level analysis. When this variable is set to **true**, please also refer to variable **timing_separate_hier_side_inputs** for further controls on reporting.

EXAMPLES

The following example shows the path reported as HyperScale side input group when both variables are set to **true**.

```
report_timing -from $side_input
****************************************
Report : timing
        -path_type full
        -delay_type max
        -max_paths 1
 ...
****************************************

  Startpoint: blk/u3/B (internal path startpoint clocked by top_CLK)
  Endpoint: blk/ff1 (rising edge-triggered flip-flop clocked by top_CLK)
  Path Group: **HyperScale_side_input**
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock top_CLK (rise edge)                0.00       0.00
  clock network delay (propagated)         0.02       0.02
  input external delay                     4.68       4.70 f
  blk/u3/B (ND2)                           0.00       4.70 f
  blk/u3/Z (ND2)                           0.64 &     5.34 r
  blk/ff1/D (FD2)                          0.00 &     5.34 r
  data arrival time                                   5.34

  clock top_CLK (rise edge)               10.00      10.00
  clock network delay (propagated)         1.16      11.16
  blk/ff1/CP (FD2)                                   11.16 r
  library setup time                      -0.85      10.31
  data required time                                 10.31
  -----------------------------------------------------------
  data required time                                 10.31
  data arrival time                                  -5.34
  -----------------------------------------------------------
  slack (MET)                                         4.97
1

report_path_group
****************************************
Report : path_group
 ...
****************************************

Path_Group     Weight   From      Through        To
--------------------------------------------------------------------------------
**HyperScale_side_input**
                1.00   { blk/u3/B ... }
                                 *              *
**async_default**
                1.00   -         -              -
**clock_gating_default**
                1.00   -         -              -
**default**     1.00   -         -              -
top_CLK         1.00   *         *              top_CLK
```

```
    top_CLK2          1.00    *          *               top_CLK2
```

SEE ALSO

```
hier_enable_analysis(3)
hier_separate_hier_side_inputs(3)
hier_enable_detailed_stub_path_timing(3)
report_timing(2)
```

hier_enable_detailed_stub_path_timing

Specifies whether to capture full details of required timing for paths ending at HyperScale stub pins (internal endpoints).

TYPE

Boolean

DEFAULT

false

DESCRIPTION

A timing path ending at a stub pin is a HyperScale-specific behavior. In the HyperScale flow, there are block-level and top-level analyses. Stub pin paths exist at the top-level analysis and are internal endpoints within a HyperScale block. Stub pins exist because block internal register-to-register paths are not fully retained at the top level. A stub pin is created when a starting register has paths leading to block output ports -- that is, starting from an interface register -- while the starting register also has timing paths leading to other registers inside the block, which makes the path shared with block internal paths. The points where a path branches to interface paths and internal paths are designated as stub pins. When a pin becomes a stub, its entire transitive fanout, including all the logic on the paths from the stub pin to the capture registers in the block, plus the clock paths of these internal capture register, are not retained in the block abstraction and therefore invisible at the top-level analysis. A stub pin is often one of the load pins of a multiload net. Topologically speaking, a stub pin is symmetric to a side input, which happens on a block input interface due to multifanin cells.

When this variable is set to **false** (the default), more compact timing data is capture for stub pin for the best runtime and memory of both block and top level timing analysis.

When this variable is set to **true**, PrimeTime performs detailed backward required time propagation and captures all the details of required time at stub pin in block level analysis, then these required times are annotated on the stub pins at top level. The reporting of thess paths end at stub pins (half of the block

internal register to register paths) is controled by another variable **timing_report_hier_stub_pin_paths**.

EXAMPLES

The following example shows the path reported in the **HyperScale_stub_default** intrinsic path group when both variables are set to **true**.

```
report_timing -path_type full_clock_expanded -from block/ffa2/Q
****************************************
Report : timing
        -path_type full_clock_expanded
        -delay_type max
        -max_paths 1
        -sort_by group
Design : SIMPLE
 ...
****************************************

  Startpoint: block/ffa2 (rising edge-triggered flip-flop clocked by SYSCLK)
  Endpoint: block/ffa4/D
              (internal path endpoint clocked by SYSCLK)
  Path Group: **HyperScale_stub_default**
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock SYSCLK (rise edge)                0.000      0.000
  clock source latency                    0.700      0.700
  CLK (in)                                0.000      0.700 r
  block/clk (BLOCK)                       0.000      0.700 r
  block/cbufa1/Y (CLKBUFX2)               0.385      1.085 r
  block/ffa2/CK (DFFX2)                   0.000      1.085 r
  block/ffa2/Q (DFFX2) <-                 0.460      1.545 f
  block/ffa4/D (DFFX2)                    0.000      1.545 f
  data arrival time                                  1.545

  clock SYSCLK (rise edge)                4.000      4.000
  clock source latency                    0.400      4.400
  CLK (in)                                0.000      4.400 r
  block/clk (BLOCK)                       0.000      4.400 r
  block/cbufa1/A (CLKBUFX2)               0.000      4.400 r
  output external delay                  -0.004      4.396
  data required time                                 4.396
  -----------------------------------------------------------
  data required time                                 4.396
  data arrival time                                 -1.545
  -----------------------------------------------------------
  slack (MET)                                        2.850


1

report_path_group
****************************************
Report : path_group
Design : SIMPLE
 ...
```

```
    ****************************************

    Path_Group     Weight   From      Through        To
    -------------------------------------------------------------
    **HyperScale_stub_default**
                   1.00   -         -              -
    **async_default**
                   1.00   -         -              -
    **clock_gating_default**
                   1.00   -         -              -
    **default**    1.00   -         -              -
    SYSCLK         1.00   *         *              SYSCLK

    1
```

SEE ALSO

hier_enable_analysis(3)
hier_enable_detailed_side_input_path_timing(3)
timing_report_hier_stub_pin_paths(3)
get_timing_paths(2)
report_timing(2)

hier_enable_distributed_analysis

Enables the HyperScale distributed analysis flow.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable the HyperScale distributed analysis flow in the PrimeTime tool.

In the distributed analysis flow, the tool analyzes the design hierarchically with single run and accurately handles the timing interfaces across hierarchical boundaries, improving the capacity and performance of hierarchical timing analysis without impacting the quality of results.

Specify the computation resources by using the **set_host_options** command. By default, HyperScale distributed analysis analyzes each block at the top level in a separate host process. To override the default partitioning of the design for analysis, use the **set_hier_config** command.

Enabling HyperScale distributed analysis affects the behavior of the **link_design**, **update_timing**, **save_session**, and reporting commands.

Because the behavior of the **link_design** command depends on whether HyperScale analysis is enabled, you must set the **hier_enable_distributed_analysis** variable before you link the design.

If the variable **hier_characterize_context_mode** is set to **constraints_only**, you cannot set the **hier_enable_distributed_analysis** variable to **true**.

You cannot set the **hier_enable_distributed_analysis** variable when the **hier_create_template_scripts** variable is set to **true**.

SEE ALSO

```
set_hier_config(2)
set_host_options(2)
start_hosts(2)
hier_characterize_context_mode(3)
hier_create_template_scripts(3)
```

hier_enable_pba_clock_latency_context

Specifies whether block context to capture path based latency for clock path entering the block, in addition to graph based latency.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable is only effective for AOCVM analysis, which is enabled by setting **timing_aocvm_enable_analysis** to **true**.

This variable applies to both the HyperScale flow and the binary context characterization flow. In HyperScale top level flow, by default, context for all the HyperScale sub-blocks are automatically captured; in flat flow, **characterize_context** and **write_context** maybe used to capture the context of target block(s).

When this variable is set to **false** (the default), the propagated clock latencies entering the block under characterization with be graph based analysis (GBA), which is accurate for block level GBA and conservative for block level PBA with the context.

When this variable is set to **true**, the propagated clock latencies is computed by performing full clock expansion back to the source and using path based analysis (PBA) to recalculate the latency. Using this context at block level PBA will further improve accuracy.

SEE ALSO

hier_enable_analysis(3)
hier_characterize_context_mode(3)
set_hier_config(2)
characterize_context(2)
report_timing(2)

hier_enable_release_resources_in_top_only_flo

Enables release of block compute resources after **update_timing** in the top-only distributed analysis flow.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to true (the default), compute resources for block partitions are released after **update_timing** in the top-only distributed analysis flow. You invoke this flow by using the command **set_hier_config -enable_top_only_flow** ...

When you set this variable to **false**, compute resources for block partitions are not released after **update_timing** in the top-only distributed analysis flow, allowing multiple timing updates in the flow.

SEE ALSO

set_hier_config(2)
update_timing(2)

hier_keep_required_time_mode

Specifies what type of required timing for stub pins are kept at the HyperScale block and loaded into HyperScale top.

TYPE

string

DEFAULT

high_fanout_stub

DESCRIPTION

The **hier_keep_required_time_mode** variable controls what type of required timing for stub pins will be kept at HyperScale block and load into HyperScale top. It can be set separately at HyperScale block and top.

- **all_stub** - Set variable to **all_stub** at block means all required timing from stub pins will be kept at block and pass to top. Set variable to **all_stub** at top means all required timing for stub pins from block will be loaded at top.
- **high_fanout_stub** - Set variable to **high_fanout_stub** at block means only required timing from high fanout stub pins will be kept at block and pass to top. Set variable to **high_fanout_stub** at top means only required timing for high fanout stub pins from block will be loaded at top.
- **internal_path_stub** - Set variable to **internal_path_stub** at block means only required timing from internal path stub pins will be kept at block and pass to top. Set variable to **internal_path_stub** at top means only required timing for internal path stub pins from block will be loaded at top.
- **no_stub** - Set variable to **no_stub** at block means no required timing from stub pins will be kept at block and pass to top. Set variable to **no_stub** at top means no timing for stub pins from block will be loaded at top.

The following table shows with the specific setting of **hier_keep_required_time_mode** variable at block and top, which set of required timing for stub pins will be available at HyperScale top:

block | all_stub | high_fanout_stub | internal_path_stub | no_stub | top | | | | | ----------+----------+----------+----------+----------+ all_stub | all stub | high fanout stub | internal path stub | no stub | ----------+----------+----------+----------+----------+ high_fanout_stub | high fanout stub | high fanout stub | no stub | no stub | ----------+----------+----------+----------+----------+ internal_path_stub |internal path stub| no stub | internal path stub | no stub | ----------+----------+----------+----------+----------+ no_stub | no stub | no stub | no stub | no stub | ----------+----------+----------+----------+----------+

SEE ALSO

save_session(2)
set_hier_config(2)

hier_merge_match_clock_with_constant

Enables clock and constant matching in HyperScale multi-instance data merge.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The default value of the **hier_merge_match_clock_with_constant** variable is **false**, which means for multi-instances blocks, PrimeTime enforces the exact matching rule for clocks across those instances in the top-level HyperScale analysis and in constraint extraction.

PrimeTime automatically performs the context data merging during HyperScale analysis when you specify more than one instance of a given block with the same configuration. However, constraint and context data merging can be performed only when the clocks are exactly matched across the instances.

Exact matching does not require that the same clocks propagate to the same block port at the top level, or the same clocks be defined on the same pins inside the different instances of the same block. Instead, it requires that those clocks physically relevant for the instances match in fundamental attributes such as clock periods and waveforms.

However, a reused block is often designed with some redundancy to be instantiated for different situations in a chip. For example, some clock network is pushed into a block as feedthrough paths for more efficient clock delivery at chip. When such a block is multiply-instantiated, certain clock ports can be intentionally tied to constants when they are not driven by global clock signals for specific modes of the instance context. At these ports, some instances have clocks while some do not; by default, this violates the clock matching rule and results in unmerged contexts or constraints across these instances.

When you set this variable to **true**, such mismatches between clock and constant across instances of the

same block are relaxed, and the merged result treats the port as having a clock because it represents a more conservative timing analysis at the block level.

Note: This variable results in merging of context and constants with a union of clocks across instances. PrimeTime does not automatically create new constraints to group the clocks by instances even if they might be truly physically exclusive across instances. This implies potentially more pessimistic block-level analysis with the merged context when comparing to the full-chip flat analysis.

SEE ALSO

save_session(2)
set_hier_config(2)
update_timing(2)

hier_modeling_version

Specifies the current modeling technique version.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable specifies the current modeling technique version. In the default version of 1.0, the verification script duplicates constraints in the instance script. There is no good way to test the instance script. When an ETM is instantiated at the top level, you must change the names of generated clocks in the constraint file so they match the names of clocks. For ILM, when a model is instantiated more than one time at the top level, you must also change the constraint file such that the names of clocks are uniquified.

If you set this variable to 2.0, the **-validate** option of the **create_ilm** and **extract_model** commands is enabled, and the generated models are automatically validated after they are created. In addition, all interface logic model (ILM) files generated are added to the pt_model_dir/ILM/design_name directory, and all extracted timing model (ETM) files are added to the pt_model_dir/ETM/output directory. These names are consistent for ILM and ETM.

In version 2.0, a test design is always created for ETM and ILM. This netlist information of the test design is in the mod_verif.v file. The constraints in the mod_verif.pt.gz file are those constraints that define the outside context (for example, the **set_input_delay** command), and the constraints in the mod_inst.pt.gz file are those constraints that are brought to the top level by the model. The mod_verif.pt.gz file sources the mod_inst.pt.gz file directly, and there are no duplicated constraints in these two files.

In version 2.0, all constraints in the mod_inst.pt.gz file are in a Tcl procedure. This procedure has two arguments: inst_name and clock_name_prefix. The constraint file needs to be sourced only one time although the model might be instantiated more than one time. The names of clocks whose sources are

inside the block and all generated clocks are uniquified. You do not need to change the constraint script to uniquify clocks. You do not need to use the **current_instance** command before sourcing this constraint file.

In version 2.0, internal nongenerated clocks are created in the mod_inst.pt.gz file for ETM. The uncertainty information of all generated clocks and internal nongenerated clocks are also added to the mod_inst.pt.gz file.

SEE ALSO

create_ilm(2)
extract_model(2)
pt_model_dir(3)

hier_override_clock_sense_from_context

Override the clock source sense if mismatch between context and constraints has been found at the pin.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, HyperScale hierarchical analysis applies the clock source sense from context.

To force the clock source sense from constraints, set the **hier_override_clock_sense_from_context** variable to **false**. This is useful when you want to forcibly apply the constrained clock sense at the pin.

SEE ALSO

hierarchy_separator

Determines how hierarchical elements of the netlist are delimited in reports and how they are searched for in selections and other commands.

TYPE

string

DEFAULT

/ (forward slash)

DESCRIPTION

This command determines how hierarchical elements of the netlist are delimited in reports and how they are searched for in selections and other commands. The choice of a separator is limited to these characters: bar (|), caret (^), at (@), dot (.), and the default of a forward slash (/).

Normally, you should accept the default, forward slash (/). However, in some cases where the hierarchy character is embedded in some names, the search engine might produce results that are not intended. Using the **hierarchy_separator** variable 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** variable to the vertical bar **|**, searching for A|B/C|D is explicit, as is A|B|C|D.

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.

This variable affects the **-auto_ignore** option of the **identify_interface_logic** command. The **-auto_ignore** option automatically determines those ports (for example, scan enable and reset ports) that should be ignored when the **identify_interface_logic** command 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 more information about creating ILMs and associated commands, see the **identify_interface_logic** man page.

SEE ALSO

```
identify_interface_logic(2)
```

imsa_save_eco_data

Specifies an additional set of attributes to be saved and restored in IMSA sessions to support physically aware ECO operations using the PrimeTime GUI.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the **save_session -only_timing_paths** command saves only the paths in the specified timing path collection, the attributes of those timing paths and their timing points, and the attributes of the associated clocks. No other attributes are saved. The **restore_session** command restores the paths and attributes to an interactive multi-scenario analysis (IMSA) session.

To enable saving and restoring a predefined additional set of attributes to support physically aware ECO operations using the PrimeTime GUI, set the **imsa_save_eco_data** variable to **true**. Then the **save_session -only_timing_paths** command saves the following additional attributes:

```
Class      Attribute Name
--------------------------
pin        max_rise_slack
pin        min_rise_slack
pin        max_fall_slack
pin        min_fall_slack
pin        max_capacitance
pin        max_transition
pin        max_fanout
```

You can also expand the set of saved and restored attributes by using **imsa_save_reporting_attributes** variable, and add or remove specific attributes by using the **set_imsa_attributes** command.

You can report the list of attributes enabled for saving by using the **set_imsa_attributes -report** command.

In DMSA mode, the value of this variable is synchronized from the master process to worker processes.

SEE ALSO

```
get_attribute(2)
get_timing_paths(2)
restore_session(2)
save_session(2)
set_imsa_attributes(2)
imsa_save_reporting_attributes(3)
```

imsa_save_reporting_attributes

Specifies an additional set of attributes to be saved and restored in IMSA sessions to support expanded reporting capabilities.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the **save_session -only_timing_paths** command saves only the paths in the specified timing path collection, the attributes of those timing paths and their timing points, and the attributes of the associated clocks. No other attributes are saved. The **restore_session** command restores the paths and attributes to an interactive multi-scenario analysis (IMSA) session.

To enable saving and restoring a predefined additional set of attributes to support expanded reporting capabilities in the IMSA session, set the **imsa_save_reporting_attributes** variable to **true**. Then the **save_session -only_timing_paths** command saves the following additional attributes:

```
Class      Attribute Name
-------------------------------------------
pin        max_rise_slack
pin        min_rise_slack
pin        max_fall_slack
pin        min_fall_slack
pin        max_capacitance
pin        max_transition
pin        actual_fall_transition_max
pin        actual_rise_transition_max
pin        annotated_rise_transition_delta_max
pin        annotated_fall_transition_delta_max
pin        constraining_max_transition
```

```
design     max_transition
design     max_capacitance
```

You can also expand the set of saved and restored attributes by using **imsa_save_eco_data** variable, and add or remove specific attributes by using the **set_imsa_attributes** command.

You can report the list of attributes enabled for saving by using the **set_imsa_attributes -report** command.

In DMSA mode, the value of this variable is synchronized from the master process to worker processes.

SEE ALSO

```
get_attribute(2)
get_timing_paths(2)
restore_session(2)
save_session(2)
set_imsa_attributes(2)
imsa_save_eco_data(3)
```

in_gui_session

This read-only variable is *true* when the graphical user interface (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 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

`gui_start(2)`
`gui_stop(2)`

input_slews_violations

This man page describes **input_slews** violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

An **input_slews** violation indicates a mismatch of input slews at a pin.

WHAT NEXT

This is an **auto_fixable** violation that HyperScale automatically resolves in subsequent iterations.

SEE ALSO

report_constraint(2)
auto_fixable_violations(3)

interactive_multi_scenario_analysis_enabled

Indicates whether the Interactive Multi-Scenario Analysis (IMSA) mode is enabled.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This read-only variable indicates whether the Interactive Multi-Scenario Analysis (IMSA) mode is enabled. IMSA is enabled by restoring a PrimeTime session saved using the **save_session** command with the **-only_timing_paths** option.

SEE ALSO

save_session(3)
restore_session(2)

lib_attributes

Describes the predefined application attributes for lib objects.

DESCRIPTION

capacitance_unit_in_farad

Type: float
Returns the unit of capacitance in the main library in farads. This attribute is read-only; you cannot change the setting.

current_unit_in_amp

Type: float
Returns the unit of current in the main library in amps. This attribute is read-only; you cannot change the setting.

default_connection_class

Type: string
Returns the default connection class string for the connection class attribute of a pin or port.

default_max_capacitance

Type: float
Returns the library default maximum capacitance design rule limit.

default_max_fanout

Type: float
Returns the library default maximum fanout design rule limit.

default_max_transition

Type: float
Returns the library default maximum transition design rule limit.

default_min_capacitance

Type: float
Returns the library default minimum capacitance design rule limit.

default_min_fanout

Type: float
Returns the library default minimum fanout design rule limit.

default_min_transition

Type: float
Returns the library default minimum transition design rule limit.

default_threshold_voltage_group

Type: string
Returns the default threshold voltage group of the cells in the library. This attribute can be defined by the library or by using the set_user_attribute command. The user-defined value takes precedence over the library-defined value. PrimePower uses this attribute to identify the threshold voltage group to which cells belong. The threshold_voltage_group library cell attribute takes precedence over the default_threshold_voltage_group library attribute if both are present when classifying cells per voltage threshold group.

extended_name

Type: string
Returns the complete, unambiguous name of the library. The extended_name of the library is the source_file_name attribute followed by a colon (:) followed by the full_name attribute. For example, the extended_name of library tech1 read in from /abc/xyz/lib1.db is /abc/xyz/lib1.db:tech1.

full_name

Type: string
Returns the name of the library. For example, the full_name of library tech1 read in from /abc/xyz/lib1.db is tech1. This name can be ambiguous because multiple libraries of the same name can be read in from different files.

has_sensitization_data

Type: boolean
Returns true if the library has sensitization data.

k_process_cell_fall

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_process_cell_rise

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_process_fall_transition

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_process_rise_transition

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_temp_cell_fall

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_temp_cell_rise

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_temp_fall_transition

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_temp_rise_transition

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_volt_cell_fall

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_volt_cell_rise

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_volt_fall_transition

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

k_volt_rise_transition

Type: float
This attribute is a nonlinear delay model (NLDM) scaling factor. For more information about this attribute, see the Library Compiler documentation.

lib_scaling_group

Type: collection
Returns a collection of libraries in the scaling library group to which the library belongs, which is set with the define_scaling_lib_group command.

min_extended_name

Type: string
Returns the full name (filename:lib_name) of the minimum library associated with the given library. This attribute is read-only; you cannot change the setting.

min_source_file_name

Type: string
Returns the full name of the minimum library associated with the given library. This attribute is read-only; you cannot change the setting.

object_class

Type: string
Returns the class of the object, which is the string "lib". You cannot set this attribute.

resistance_unit_in_ohm

Type: float
Returns the unit of resistance in the main library in ohms. This attribute is read-only; you cannot change the setting.

source_file_name

Type: string
Returns the name of the file from which the library was read. For example, the source_file_name of library tech1 read in from /abc/def/lib1.db is /abc/def/lib1.db.

time_unit_in_second

Type: float
Returns the unit of time in the main library in seconds. This attribute is read-only; you cannot change the setting.

voltage_unit_in_volt

Type: float
Returns the unit of voltage in the main library in volts. This attribute is read-only; you cannot change the setting.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

lib_cell_attributes

Describes the predefined application attributes for lib_cell objects.

DESCRIPTION

always_on

Type: boolean
Returns true if the library cell is a UPF always-on cells.

area

Type: float
Returns the area of the library cell.

base_name

Type: string
Returns the name of the library cell. For example, the base_name of library cell tech1/AN2 is AN2.

disable_timing

Type: boolean
Returns true if the timing for the library cell has been specified to be disabled using the set_disable_timing command.

dont_touch

Type: boolean
Returns true if the library cell is excluded from changes. Values are undefined by default. Instances of library cells with the dont_touch attribute set to true are not modified or replaced by ECO fixing commands (fix_eco_drc, fix_eco_power, and fix_eco_timing). This attribute is set by the set_dont_touch command.

dont_use

Type: boolean
Returns true if the library cell is excluded from the target library during ECO fixing. This attribute is set by the set_dont_use command.

full_name

Type: string
Returns the fully qualified name of the library cell. This is the name of the library followed by the library cell name. For example, the full_name of library cell AN2 in library tech1 is tech1/AN2.

function_id

Type: string
Returns the name of the function that is created by Library Compiler.

has_multi_ground_rails

Type: boolean
Returns true if the library cell has multiple ground rails.

has_multi_power_rails

Type: boolean
Returns true if the library cell has multiple power rails.

has_rail_specific_power_tables

Type: boolean
Returns true if the library cell has multiple tables attached to rails.

is_black_box

Type: boolean
Returns true if there is no reference library cell from any of the link libraries.

is_combinational

Type: boolean
Returns true if the library cell has no sequential timing arcs. See also the report_lib command.

is_case_sequential_propagation

Type: boolean
Returns true if the library cell is enabled for sequential case propagation by the set_case_sequential_propagation command.

is_fall_edge_triggered

Type: boolean
Returns true if the library cell is used as a falling-edge-triggered flip-flop.

is_instantiated

Type: boolean
Returns true if the library cell is instantiated in the current design.

is_integrated_clock_gating_cell

Type: boolean

Returns true if the cell's reference is not linked to a library cell or design. This attribute is read-only.

is_isolation

Type: boolean
Returns true if the library cell is a UPF isolation cell.

is_level_shifter

Type: boolean
Returns true if the library cell is a UPF level shifter cell.

is_macro_switch

Type: boolean
Returns true if the library cell is defined as a fine-grain switch cell in the library.

is_memory_cell

Type: boolean
Returns true if the library cell is classified in the memory power group.

is_mux

Type: boolean
Returns true if the library cell is a multiplexer.

is_negative_level_sensitive

Type: boolean
Returns true if the library cell is used as a negative level-sensitive latch.

is_pad_cell

Type: boolean
Returns true if the library cell is a pad cell.

is_pll_cell

Type: boolean
Returns true if the library cell is used as a phase-locked loop (PLL) cell.

is_positive_level_sensitive

Type: boolean
Returns true if the library cell is used as a positive level-sensitive latch.

is_retention

Type: boolean
Returns true if the library cell is a UPF retention cell.

is_rise_edge_triggered

Type: boolean
Returns true if the library cell is used as a rising-edge-triggered flip-flop.

is_sequential

Type: boolean
Returns true if the library cell has at least one sequential timing arc. See also the report_lib command.

is_three_state

Type: boolean
Returns true if the library cell is a three-state device.

lib_pg_pin_info

Type: collection
Returns a collection of lib_pg_pin_info objects with these attributes: pin_name, type, voltage.

mog_func_id

Type: string
Returns the name of the cell's function that is created by Library Compiler for multiple output gate (MOG) cells.

number_of_pins

Type: integer
Returns the number of pins on the library cell.

object_class

Type: string
Returns the class of the object, which is "lib_cell". You cannot set this attribute.

power_cell_type

Type: string
Returns the predefined power group to which the cell belongs. Use the set_user_attribute command to set the attribute to one of these predefined power groups: clock_network, register, combinational, sequential, memory, io_pad, or black_box. You cannot use this attribute to assign a cell to a user-defined power group. The power_cell_type cell attribute overrides the power_cell_type library cell attribute. PrimePower uses this attribute in both averaged and time-based power analysis modes.

threshold_voltage_group

Type: string
Returns the threshold voltage group of the library cell. The attribute can be defined by the library or by using the set_user_attribute command. The user-specified value takes precedence over the library-defined value. If the attribute is undefined, it returns "No Default Threshold".

timing_model_type

Type: string

Returns the timing model type of the library cell. The possible values are ITS (interface timing specification), quick timing model, extracted, and none (normal library model).

user_function_class

Type: string
Returns the functional behavior of the library cell.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

lib_pg_pin_info_attributes

Describes the predefined application attributes for lib_pg_pin_info objects.

DESCRIPTION

pin_name

Type: string
Returns the pin name.

type

Type: string
Returns primary_power or primary_ground.

voltage

Type: float
Returns the pin voltage.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

lib_pin_attributes

Describes the predefined application attributes for lib_pin objects.

DESCRIPTION

base_name

Type: string
Returns the leaf name of the library cell pin. For example, the base_name of tech1/AN2/Z is Z.

clock

Type: boolean
Returns true when a clock attribute is attached to the library pin in the library definition.

connection_class

Type: string
Returns the connection class string for the pin.

direction

Type: string
Returns the direction of the pin. Value can be in, out, inout, or internal.

disable_timing

Type: boolean
Returns true if the library pin has been specified to be disabled using the set_disable_timing command.

drive_resistance_fall

Type: float
Returns the linear drive resistance for falling delays of the library pin.

drive_resistance_rise

Type: float
Returns the linear drive resistance for rising delays of the library pin.

driver_waveform_fall

Type: string
Returns the type of driver waveform for the library pin, either ramp or standard.

driver_waveform_rise

Type: string
Returns the type of driver waveform for the library pin, either ramp or standard.

fanout_load

Type: float
Returns the fanout load value of the library pin. This value is used in computing max_fanout design rule cost.

full_name

Type: string
Returns the fully qualified name of a library cell pin. This is the name of the library followed by the library cell name followed by a pin name. For example, the full_name of pin Z on library cell AN2 in library tech1 is tech1/AN2/Z.

has_ccs_noise_above_high

Type: boolean
Returns true if CCS noise information is present for a pin in a library.

has_ccs_noise_above_low

Type: boolean
Returns true if CCS noise information is present for a pin in a library.

has_ccs_noise_below_high

Type: boolean
Returns true if CCS noise information is present for a pin in a library.

has_ccs_noise_below_low

Type: boolean
Returns true if CCS noise information is present for a pin in a library.

has_ccs_receiver_fall

Type: boolean
Returns true if CCS information is present for a pin in a library, for receiver fall analysis.

has_ccs_receiver_rise

Type: boolean
Returns true if CCS information is present for a pin in a library, for receiver rise analysis.

is_async_pin

Type: boolean
Returns true if the library pin is an asynchronous preset/clear pin.

is_clear_pin

Type: boolean
Returns true if the library pin is an asynchronous clear pin.

is_clock_pin

Type: boolean
Returns true if at least one instance of that clock pin exists that has the is_clock_pin attribute equal to true.

is_data_pin

Type: boolean
Returns true if at least one instance of that data pin exists that has the is_data_pin attribute equal to true.

is_fall_edge_triggered_clock_pin

Type: boolean
Returns true if the library pin is used as a falling-edge-triggered flip-flop clock pin.

is_fall_edge_triggered_data_pin

Type: boolean
Returns true if the library pin is used as a falling-edge-triggered flip-flop data pin.

is_mux_select_pin

Type: boolean
Returns true if the library pin is a select pin of a multiplexer device.

is_negative_level_sensitive_clock_pin

Type: boolean
Returns true if the library pin is used as a negative level-sensitive latch clock pin.

is_negative_level_sensitive_data_pin

Type: boolean
Returns true if the library pin is used as a negative level-sensitive latch data pin.

is_pad

Type: boolean
Returns true if the library pin is a pad. See the Library Compiler documentation.

is_pll_feedback_pin

Type: boolean

Returns true if the library pin is a feedback pin of a phase locked loop (PLL) cell.

is_pll_output_pin

Type: boolean
Returns true if the library pin is an output pin of a phase locked loop (PLL) cell.

is_pll_reference_pin

Type: boolean
Returns true if the library pin is a reference pin of a phase locked loop (PLL) cell.

is_positive_level_sensitive_clock_pin

Type: boolean
Returns true if the library pin is used as a positive level-sensitive latch clock pin.

is_positive_level_sensitive_data_pin

Type: boolean
Returns true if the library pin is used as a positive level-sensitive latch data pin.

is_preset_pin

Type: boolean
Returns true if the library pin is an asynchronous preset pin.

is_rise_edge_triggered_clock_pin

Type: boolean
Returns true if the library pin is used as a rising-edge-triggered flip-flop clock pin.

is_rise_edge_triggered_data_pin

Type: boolean
Returns true if the library pin is used as a rising-edge-triggered flip-flop data pin.

is_three_state

Type: boolean
Returns true if the library pin is a three-state driver.

is_three_state_enable_pin

Type: boolean
Returns true if the library pin is an enable pin of a three-state device.

is_three_state_output_pin

Type: boolean
Returns true if the library pin could output a three-state signal.

is_unbuffered

Type: boolean
Returns true if the library pin is unbuffered. See the Library Compiler documentation.

load_of_pin_capacitance

Type: float
Returns the capacitance as specified by the Liberty file by the capacitance attribute in the pin object class.

max_capacitance

Type: float
Returns the maximum capacitance design rule limit for the library pin.

max_fanout

Type: float
Returns the maximum fanout design rule limit for the library pin.

max_transition

Type: float
Returns the maximum transition time design rule limit for the library pin.

min_capacitance

Type: float
Returns the minimum capacitance design rule limit for the library pin.

min_fanout

Type: float
Returns the minimum fanout design rule limit for the library pin.

min_transition

Type: float
Returns the minimum transition time design rule limit for the library pin.

object_class

Type: string
Returns the class of the object, which is a constant equal to lib_pin. You cannot set this attribute.

original_pin

Type: string
Returns the original pin name defined in the block netlist that is used to generate an extracted timing model (ETM).

pin_capacitance

Type: float

Returns the capacitance of the library pin.

pin_capacitance_max_fall

Type: float
Returns the maximum fall capacitance of the library pin. This attribute is read-only; you cannot change the setting.

pin_capacitance_max_rise

Type: float
Returns the maximum rise capacitance of the library pin. This attribute is read-only; you cannot change the setting.

pin_capacitance_min_fall

Type: float
Returns the minimum fall capacitance of the library pin. This attribute is read-only; you cannot change the setting.

pin_capacitance_min_rise

Type: float
Returns the minimum rise capacitance of the library pin. This attribute is read-only; you cannot change the setting.

pin_direction

Type: string
Returns the direction of a pin. Allowed values are in, out, inout, or unknown. This attribute is read-only; you cannot change the settings.

rc_input_threshold_pct_fall

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. Specifies the value obtained from the library to which the pin belongs.

rc_input_threshold_pct_rise

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. Specifies the value obtained from the library to which the pin belongs.

rc_output_threshold_pct_fall

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. Specifies the value obtained from the library to which the pin belongs.

rc_output_threshold_pct_rise

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. Specifies the value obtained from the library to which the pin belongs.

rc_slew_derate_from_library

Type: float
Returns the slew derating factor that the tool uses to calculate delays and transition times. Specifies the value obtained from the library to which the pin belongs.

rc_slew_lower_threshold_pct_fall

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. Specifies the value obtained from the library to which the pin belongs.

rc_slew_lower_threshold_pct_rise

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. Specifies the value obtained from the library to which the pin belongs.

rc_slew_upper_threshold_pct_fall

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. Specifies the value obtained from the library to which the pin belongs.

rc_slew_upper_threshold_pct_rise

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. Specifies the value obtained from the library to which the pin belongs.

si_has_immunity_above_high

Type: boolean
Returns true if NLDM noise immunity information is present for a pin in a library.

si_has_immunity_above_low

Type: boolean
Returns true if NLDM noise immunity information is present for a pin in a library.

si_has_immunity_below_high

Type: boolean
Returns true if NLDM noise immunity information is present for a pin in a library.

si_has_immunity_below_low

Type: boolean
Returns true if NLDM noise immunity information is present for a pin in a library.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

lib_timing_arc_attributes

Describes the predefined application attributes for lib_timing_arc objects.

DESCRIPTION

from_lib_pin

Type: collection
Returns a collection containing the from library pin of the library timing arc.

has_ccs_driver_fall

Type: boolean
Returns true if the library timing arc has CCS timing driver information for fall analysis.

has_ccs_driver_rise

Type: boolean
Returns true if the library timing arc has CCS timing driver information for rise analysis.

has_ccs_noise_above_high

Type: boolean
Returns true if CCS noise information is present in the timing arc object in a library.

has_ccs_noise_above_low

Type: boolean
Returns true if CCS noise information is present in the timing arc object in a library.

has_ccs_noise_below_high

Type: boolean
Returns true if CCS noise information is present in the timing arc object in a library.

has_ccs_noise_below_low

Type: boolean
Returns true if CCS noise information is present in the timing arc object in a library.

has_ccs_receiver_fall

Type: boolean
Returns true if CCS receiver information is present in the timing arc object in a library, for fall analysis.

has_ccs_receiver_rise

Type: boolean
Returns true if CCS receiver information is present in the timing arc object in a library, for rise analysis.

is_disabled

Type: boolean
Returns true if the library timing arc is disabled.

is_user_disabled

Type: boolean
Returns true if the library timing arc is disabled by the set_disable_timing command.

mode

Type: string
Returns the mode string of the library timing arc.

object_class

Type: string
Returns the class of the object, which is the string "lib_timing_arc". You cannot set this attribute.

sdf_cond

Type: string
Returns a string representing the SDF condition of the library timing arc.

sense

Type: string
Returns a string representing the sense of the library timing arc.

si_has_immunity_above_high

Type: boolean
Returns true if NLDM noise immunity information is present in the timing arc object in a library.

si_has_immunity_above_low

Type: boolean
Returns true if NLDM noise immunity information is present in the timing arc object in a library.

si_has_immunity_below_high

Type: boolean
Returns true if NLDM noise immunity information is present in the timing arc object in a library.

si_has_immunity_below_low

Type: boolean
Returns true if NLDM noise immunity information is present in the timing arc object in a library.

si_has_iv_above_high

Type: boolean
Returns true if the library timing arc contains output steady-state information in the form of I/V relationships (polynomials or tables).

si_has_iv_above_low

Type: boolean
Returns true if the library timing arc contains output steady-state information in the form of I/V relationships (polynomials or tables).

si_has_iv_below_high

Type: boolean
Returns true if the library timing arc contains output steady-state information in the form of I/V relationships (polynomials or tables).

si_has_iv_below_low

Type: boolean
Returns true if the library timing arc contains output steady-state information in the form of I/V relationships (polynomials or tables).

si_has_propagation_above_high

Type: boolean
Returns true if the library timing arc contains information about how bumps present at the arc input are propagated across the arc to the output.

si_has_propagation_above_low

Type: boolean
Returns true if the library timing arc contains information about how bumps present at the arc input are propagated across the arc to the output.

si_has_propagation_below_high

Type: boolean
Returns true if the library timing arc contains information about how bumps present at the arc input are propagated across the arc to the output.

si_has_propagation_below_low

Type: boolean
Returns true if the library timing arc contains information about how bumps present at the arc input are propagated across the arc to the output.

si_has_resistance_above_high

Type: boolean
Returns true if the library timing arc contains output steady-state information in the form of simple steady drive resistance.

si_has_resistance_above_low

Type: boolean
Returns true if the library timing arc contains output steady-state information in the form of simple steady drive resistance.

si_has_resistance_below_high

Type: boolean
Returns true if the library timing arc contains output steady-state information in the form of simple steady drive resistance.

si_has_resistance_below_low

Type: boolean
Returns true if the library timing arc contains output steady-state information in the form of simple steady drive resistance.

to_lib_pin

Type: collection
Returns a collection containing the to library pin of the library timing arc.

when

Type: string
Returns a string representing the when string of the library timing arc.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

library_mapping_violations

This man page describes **library_mapping** violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

A **library_mapping** violation indicates differences in the libraries being used. The use of libraries includes linking the design, selecting operating condition, and defining the scaling library group. A mismatch in libraries used between block- and top-level analysis can cause different timing results and therefore need to be fixed.

Timing constraint commands referring to libraries include (and not limited to) **set_min_library**, **define_scaling_lib_group**, **set_operating_condition**, **set_wire_load_model**, also the variables such as **link_path**.

Note that **library_mapping** violations might also result in other violations such as operating conditions and timing derates, because they can refer to libraries that are different between block and top, and result in violations reported for those other checks also.

WHAT NEXT

This is a **non_auto_fixable** violation. You must resolve the difference in libraries. Consistent usage and settings of library related constraints are required to ensure consistent results for the timing analysis.

Consistent settings and usage of libraries between the block and top levels can also resolve other types of HyperScale boundary constraint violations, such as difference in operating conditions and library-cell derating.

EXAMPLES

The following verbose report shows a simple difference of the top level enabling CCS calculation while the block level does not use CCS.

```
pt_shell> report_constraint -boundary_check_include {library_mapping} \
```

```
          -all_violators -verbose

****************************************
Report : constraint
        -verbose
 ...
****************************************

HyperScale constraints report

Constraint: library_mapping

Instance   Type            Block                                Top
-------------------------------------------------------------------------------------------------
blk1       Main Library    /my/libraries/A_lib.db:nldm_lib     /remote/libraries/ccs_1.1.db:ccs_lib
```

SEE ALSO

define_scaling_lib_group(2)
report_constraint(2)
set_min_library(2)
set_operating_conditions(2)
set_wire_load_model(2)
link_path(3)
non_auto_fixable_violations(3)
search_path(3)

library_pg_file_pattern

Specifies the file name pattern for the power and ground (PG) Tcl side files for library PG conversion and update.

TYPE

string

DEFAULT

""

DESCRIPTION

Use this variable to specify the file name pattern for the PG Tcl side files for library PG conversion and update. By setting the variable to a valid file name pattern, the tool finds the associated PG Tcl side file for the logic libraries provided. At library loading time, the tool performs on-the-fly PG updates on the in-memory databases. With this on-the-fly library PG conversion and update capability, the tool relaxes the requirement of complete PG pin library for UPF specifications.

As default, the variable is set to "", which means that there is no PG Tcl side file, unless specified.

For advanced users, string substitution can be used for finding PG Tcl side file (limit one occurrence per pattern):

- Use pattern "__DIR__" for path to .db dir
- Use pattern "__FILE__" for leaf file name for .db

There can be one Tcl for all databases, one Tcl per database, or one Tcl per a group of databases. For example:

1. One Tcl for all databases:

- At the current directory

  ```
  set library_pg_file_pattern "libpg_sidefile.pg"
  ```

- At different location

  ```
  set library_pg_file_pattern "/my_dir/libpg_sidefile.pg"
  ```

2. One Tcl per database:

   - At the same location as the original database files

     ```
     set library_pg_file_pattern "%d/%f.pg"
     ```

   - At a directory called "pg_sidefiles" under the same directory as original the original database

     ```
     set library_pg_file_pattern "%d/pg_sidefiles/%f.pg"
     ```

   - At a different location called "/my_dir"

     ```
     set library_pg_file_pattern "/my_dir/%d/%f.pg"
     ```

3. One Tcl for a group of databases at the same location as the database files

   ```
   set library_pg_file_pattern "%d/libpg_sidefile.pg"
   ```

Note: You must set this variable before loading the libraries.

This on-the-fly database updates is disabled in the power domain backward compatibility mode (**set power_domains_compatibility true**).

SEE ALSO

power_domains_compatibility(3)

link_allow_design_mismatch

Controls the behavior of the link design when pin mismatch between instance and reference occur.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether design linking succeeds when pin mismatches between instance and reference occur. By default, linking fails if there are pin mismatches between the instance and reference. For example, when a pin exists in the instance but does not exist in the library, link issues an error and fails.

If you set this variable to **true**, the extra pin is ignored, and design linking continues. This allows you to gather useful information even if part of a design is missing.

Common causes of mismatches include:

1. A pin has different directions in instance and reference.
2. A pin of instance does not exist in reference.
3. A bus has different widths in instance and reference.

When a mismatch occurs, the reference always take precedence.

- In case 1, the direction of the pin in the linked design is from the reference.
- In case 2, the pin is ignored and does not exist in the reference.

- In case 3, all extra bits in the instance are ignored, and the bus width in the linked design is the same as the bus width from the reference.

Note that for bus width mismatches, the least significant bit of instance is mapped to least significant bit of the reference; all extra bits of the instance are ignored, and all extra bits of the reference are dangling.

To report mismatches found during link, use the **report_design_mismatch** command.

SEE ALSO

```
link_design(2)
report_design_mismatch(2)
```

link_create_black_boxes

Enables design linking to automatically convert unresolved references into black boxes.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, this variable is set to **true**, and design linking automatically converts each unresolved reference into a black box, an empty cell with no timing arcs. The result is a completely linked design on which you can analysis.

If you set this variable to **false**, unresolved references remain unresolved, and most analysis commands cannot work.

SEE ALSO

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

link_force_case

Controls the case sensitivity behavior of the link design command.

TYPE

string

DEFAULT

check_reference

DESCRIPTION

This variable controls the case-sensitive or insensitive behavior of the **link_design** command. Allowed values are the default of **check_reference**, **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: Design Compiler 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 that can cause confusion.

3. Setting the **link_force_case** variable to **case_insensitive** can decrease the performance of the

link_design command.

SEE ALSO

link_design(2)
link_create_black_boxes(3)

link_keep_cells_with_pg_only_connection

Keeps cells that are connected to only power or ground (PG) nets after design linking.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether to keep cells that are connected only to power or ground (PG) nets after design linking. This variable is ignored if the **link_keep_unconnected_cells** variable is set to **true**.

- **false** (the default) - Deletes cells that are connected only to PG nets.
- **true** - Keeps cells that are connected only to PG nets.

You must set this variable before you load designs and libraries into PrimeTime.

SEE ALSO

```
link_design(2)
link_keep_unconnected_cells(3)
```

link_keep_pg_connectivity

Enables the Verilog PG flow, which reads in Verilog PG netlist and derives the rail connectivity data from that netlist.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the PrimeTime tool uses UPF-specified power intent and ignores PG connectivity information in Verilog netlists. However, you can optionally have the tool to derive PG connectivity from a PG Verilog netlist, in which case there is no need to load UPF files.

To do this, set the **link_keep_pg_connectivity** variable to **true** before you read in any design information. When you read in the netlist, the tool derives the UPF power intent from the PG netlist data and implicitly updates the design database as if it were running UPF commands like **create_power_domain**, **create_supply_port**, **create_supply_net**, and **connect_supply_net**. After you link the design, you can set the supply voltages by using the **set_voltage** command, just like the UPF flow.

This PG netlist flow works with design data generated in either the UPF-prime flow or the golden UPF flow. It supports PrimeTime features such as timing analysis, voltage scaling, power switches, and ECOs. Note that the PG netlist flow requires all the netlists to have PG information; it does not accept mixture of PG and non-PG netlists.

Loading the UPF file might be necessary for using the **extract_model** command in combination with the **set_port_attributes** and **set_related_supply_net** commands.

SEE ALSO

connect_supply_net(2)
create_power_domain(2)
create_power_switch(2)
create_supply_port(2)

link_keep_unconnected_cells

Keeps unconnected cells after design linking.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether to keep unconnected cells after design linking.

- **false** (the default) - Discards unconnected cells. By default, the tool also discards cells that are connected to only power/ground (PG) nets; to keep cells that are connected to only PG nets, set the **link_keep_cells_with_pg_only_connection** to **true**.
- **true** - Keeps unconnected cells. With this setting, the tool also keeps cells that are connected to PG nets, regardless of the **link_keep_cells_with_pg_only_connection** variable setting.

You must set this variable before you load designs and libraries into PrimeTime.

SEE ALSO

```
link_design(2)
link_keep_cells_with_pg_only_connection(3)
```

link_path

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

TYPE

list

DEFAULT

*

DESCRIPTION

This variable 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 that you specify.

The **link_path** variable can contain three 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 the pt_shell while trying to resolve references. Designs are searched in the order in which they were read.

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

The libraries that are specified by the **link_path** variable are loaded in parallel with Verilog files during the **read_verilog** command. For best runtime performance, set the **search_path** and **link_path** variables before you run the **read_verilog** command.

SEE ALSO

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

link_path_per_instance

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

TYPE

list

DEFAULT

(empty)

DESCRIPTION

This variable, which takes effect only if set before 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 that 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 preceding example:

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.

SEE ALSO

```
link_design(2)
link_path(3)
link_path_per_instance(3)
```

lp_default_ground_pin_name

Specifies the default ground pin name for creating default PG pins for the library cells in non-PG pin libraries.

TYPE

string

DEFAULT

""

DESCRIPTION

Use this variable to specify the default ground pin name for default PG pin creation. This variable is also used for ground pin name for database conversion for legacy rail_connection libraries. One default power and one default ground PG pins are created for the library cells in non-PG pin libraries, which are performed on the in-memory databases by the tool at library loading time. With this capability, the tool relaxes the requirement of PG pin library for UPF specifications.

The name is used for pg_pin of library cells and voltage_map defined at library level. For library cells from PG pin libraries, the variable have no impact. The default PG pin creation isl only triggered when the PG Tcl side file is not provided (specified by the **library_pg_file_pattern** variable). UPF explicit connections (using the **connect_supply_net** command) are not allowed on default PG pins created based on the default PG pin name variables.

The default of this variable is empty "", which means that there is no default PG pin added unless specified. To enable default PG pin creation, set the **lp_default_power_pin_name** variable to a valid name.

Note: You must set this variable before loading libraries.

Database updates are disabled in power domain backward compatibility mode (when you set the **power_domains_compatibility** variable to **true**).

SEE ALSO

```
connect_supply_net(2)
library_pg_file_pattern(3)
lp_default_power_pin_name(3)
power_domains_compatibility(3)
```

lp_default_power_pin_name

Specifies the default power pin name for creating default PG pins for the library cells in non-PG pin libraries.

TYPE

string

DEFAULT

""

DESCRIPTION

Use this variable to specify the default power pin name for default PG pin creation. One default power and one default ground PG pins is created for the library cells in non-PG pin libraries, which is performed on the in-memory databases by the tool at the time the library is loaded. With this capability, the tool relaxes the requirement of PG pin library for UPF specifications.

The name is used for pg_pin of library cells and voltage_map defined at library level. For library cells from PG pin libraries, the variable have no impact. The default PG pin creation is triggered only when the PG Tcl side file is not provided (specified by the **library_pg_file_pattern** variable). UPF explicit connections (using the **connect_supply_net**) is not allowed on default PG pins created based on the default PG pin name variables.

The default of this variable is empty "", which means that there is no default PG pin added unless specified. To enable default PG pin creation, you must set the **lp_default_ground_pin_name** variable to a valid name.

Note: You must set this variable before loading the library.

The database updates are disabled in power domain backward compatibility mode (Use the **set power_domains_compatibility TRUE** command).

SEE ALSO

```
connect_supply_net(2)
library_pg_file_pattern(3)
lp_default_ground_pin_name(3)
power_domains_compatibility(3)
```

merge_model_allow_generated_clock_renamin

Enables or disables forcing merging ETM models by resolving conflicting generated clocks.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When you merge an extracted timing model (ETM), generated clocks in each ETM model with the same name should have the same definition. By default, if you use the **merge_models** command to merge ETMs with generated clocks of the same name but different definitions, the tool issues an error message, and the model merging fails.

When you set this variable to **true**, this issue is resolved by renaming the generated clock to include the corresponding mode name as suffixes.

SEE ALSO

`merge_model_ignore_pin_function_check(3)`

merge_model_ignore_pin_function_check

Enables or disables forcing merging ETM models to ignore function attributes.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When you merge ETM models, the tool expects either a **true** or a **false** in the **function** attribute. If you use the **merge_models** command to merge ETM models with non-constant Boolean functions, the tool issues an error, and model merging fails. If you set this variable to **true**, the models are forced to merge, ignoring the **function** attribute.

SEE ALSO

```
merge_models(2)
merge_model_allow_generated_clock_renaming(3)
```

mode_attributes

Describes the predefined application attributes for mode objects.

DESCRIPTION

name

Type: string
Returns the name of the mode object.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

model_validation_capacitance_tolerance

Specifies the absolute error tolerance for capacitance data.

TYPE

list

DEFAULT

{0.001 0.001}

DESCRIPTION

Specifies the absolute error tolerance for capacitance data. If this is used in conjunction with the **-percent_tolerance** option, both tolerances must be exceeded to cause a FAIL. For information about other absolute tolerances, see the **model_validation_timing_tolerance** man page.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
model_validation_timing_tolerance(3)
```

model_validation_check_design

Checks the design constraints and settings when auto model validation fails

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If you set this variable to **true**, the tool tries to find design constraints and settings that are not recommended for creating and validating models.

If you set this variable to **false**, no checking is performed.

This variable is effective only in the automatic model validation flow.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
```

model_validation_ignore_pass

Excludes all passing comparisons; therefore, fails are only displayed if this variable is set to its default of **true**.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Excludes all passing comparisons; therefore, fails are only displayed if this variable is set to its default of **true**.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

create_ilm(2)
extract_model(2)
hier_modeling_version(3)

model_validation_output_file

Specifies the name of the file that stores the model validation results.

TYPE

string

DEFAULT

verif.out

DESCRIPTION

Specifies where the model validation results should be written. If this variable is empty, the results are printed at the standard output. If no path is given in this name, this file is created in the model_validation subdirectory under the directory specified by the **pt_model_dir** variable.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
pt_model_dir(3)
```

model_validation_pba_clock_path

Enables path-based analysis on clock portion of mismatched paths.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, path-based analysis in model validation is only performed on the data portion of mismatched paths. When you set this variable to **true**, path-based analysis is performed on the clock portion too. This can help resolving mismatched paths.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
```

model_validation_percent_tolerance

Specifies the absolute error tolerance for time data during model validation.

TYPE

list

DEFAULT

{0.00 0.00}

DESCRIPTION

This option is similar to the **model_validation_timing_tolerance** variable except that it uses the percent relative error instead of an absolute error. The exception is that slack data ignores this variable and uses only the **model_validation_timing_tolerance** variable. If this variable is specified in conjunction with either the **model_validation_timing_tolerance** or **model_validation_capacitance_tolerance** variable, both tolerances must be exceeded to cause a FAIL.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
model_validation_capacitance_tolerance(3)
```

```
model_validation_timing_tolerance(3)
```

model_validation_reanalyze_max_paths

Specifies the maximum number of mismatched paths that are analyzed again.

TYPE

integer

DEFAULT

1000

DESCRIPTION

During automatic model validation, some mismatched paths after graph-based analysis are reselected to do path-based analysis. This variable specifies the maximum number of selected paths. The smaller the number, the faster the validation occurs; however, more unresolved mismatches might occur.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
```

model_validation_report_split

Enables line-splitting in the validation report.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Allows line-splitting in a report. This is most useful for performing diffs on previous scripts or for postprocessing the script.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
```

model_validation_save_session

Saves a session during model validation that constrains the testing design of the model.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When the **model_validation_save_session** variable is set to **true** (the default), a session is saved that constrains the testing design of the model.

If you set this variable to **false**, no session is saved.

This variable is effective only in the automatic model validation flow.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
```

model_validation_section

Specifies a list of data sections to be included in the comparison.

TYPE

string

DEFAULT

"" (empty)

DESCRIPTION

Specifies a list of data sections to be included in the validation; only those sections are validated. By default, all sections are validated. If it is not empty, defined sections are validated. The following list describes the keywords that are accepted in the section_list as valid:

- **slack** - Specifies that only worst-case slacks are to be validated.
- **transition_time** - Specifies that only actual transition times on ports are to be validated.
- **capacitance** - Specifies that only actual capacitances on ports are to be validated.
- **design_rules** - Specifies that only design rules on ports are to be validated.
- **missing_arcs** - Specifies that only arcs in the reference timing file that are missing from the compare timing file are to be included in the report.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

create_ilm(2)
extract_model(2)
hier_modeling_version(3)

model_validation_significant_digits

Specifies the number of digits to the right of the decimal point that are to be reported in the validation results.

TYPE

integer

DEFAULT

3

DESCRIPTION

Allowed values are 0-13. The default value is 3.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
```

model_validation_sort_by_worst

Specifies that the output is to be sorted from worst to best.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Specifies that the output is to be sorted from negative to positive, with the worst failure first. After the FAIL section, all PASS lines are sorted alphabetically within each path attribute.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
```

model_validation_timing_tolerance

Specifies the absolute error tolerance for time data during model validation.

TYPE

list

DEFAULT

{0.01 0.025}

DESCRIPTION

Specifies the absolute error tolerance for time data during model validation. Each tolerance is a list of either one or two floating point numbers. All tolerances are inclusive, and the sign of the values has no effect. The first value is the optimistic tolerance and the second value is the pessimistic tolerance. If there is only one number it serves to specify both the pessimistic and optimistic tolerances. If this is used in conjunction with the **model_validation_percent_tolerance** variable, both tolerances must be exceeded to cause a FAIL. For other absolute tolerances, see the **model_validation_capacitance_tolerance** man page.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the **compare_interface_timing** command is not affected by this variable.

SEE ALSO

create_ilm(2)
extract_model(2)

```
hier_modeling_version(3)
model_validation_percent_tolerance(3)
```

model_validation_verbose

Specifies that detailed debugging of mismatched paths is performed.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Specifies that a detailed debugging of mismatched paths is performed. If you set this variable to **true**, a detailed debugging of mismatched paths is performed, which generates a report that shows why those paths are mismatched. The name of the report is the name of the output file specified by the **model_validation_output_file** variable with the .detail extension.

For ILMs, the tolerance of delay and slews is one-tenth of the tolerance specified by the **model_validation_timing_tolerance** variable, and the tolerance of capacitance is the same with the **model_validation_capacitance_tolerance** variable. If either of these tolerances are violated for a stage, the detailed information of this stage is printed; however, only the first stage violating the tolerances is printed.

For ETMs, since paths from the model are generally simple, the **report_etm_arc** style report is printed for each mismatched arc and stage information is not compared.

If it is found that the topologies of paths from the netlist and the model are different with each other, another detailed debugging is performed: the exactly same path from the model is selected from the netlist and compared against with the original path selected from the netlist. This debugging can help you understand why the **create_ilm** or **extract_model** command picks a different critical path.

This variable is effective only in the automatic model validation flow, such as when the output from either the **create_ilm** or **extract_model** command is used with the **-validate** option. The output of the

compare_interface_timing command is not affected by this variable.

EXAMPLES

The following examples shows that the mismatch is because an aggressor that is not screened in the block gets screened when the interface logic model (ILM) is used.

```
Netlist path:

  Startpoint: IN (input port clocked by CLK)
  Endpoint: ff1 (rising edge-triggered flip-flop clocked by CLK')
  Path Group: CLK
  Path Type: max

  Point                        Fanout    Cap      Trans       Incr       Path
-------------------------------------------------------------------------------
  clock CLK (rise edge)                                       0.00       0.00
  clock network delay (propagated)                            0.00       0.00
  input external delay                                        0.20       0.20 f
  IN (in)                                          0.00       0.00       0.20 f
  IN (net)                        1     1.00
  U1/A (ND2)                                       0.00       0.00       0.20 f
  U1/Z (ND2)                                       0.17       0.67 &     0.87 r
  n1 (net)                        1     1.23
  ff1/D (FD1)                                      0.18       0.01 &     0.88 r
  data arrival time                                                      0.88

  clock CLK' (rise edge)                                      0.60       0.60
  clock source latency                                        0.00       0.60
  CLK (in)                                         0.00       0.00       0.60 f
  CLK (net)                       3     3.00
  U2/A (IV)                                        0.00       0.00       0.60 f
  U2/Z (IV)                                        0.14       0.52       1.13 r
  n2 (net)                        1     1.00
  ff1/CP (FD1)                                     0.14       0.00       1.13 r
  library setup time                                         -0.80       0.33
  data required time                                                     0.33
-------------------------------------------------------------------------------
  data required time                                                     0.33
  data arrival time                                                     -0.88
-------------------------------------------------------------------------------
  slack (VIOLATED)                                                      -0.55


The dominant exceptions are:
        None

Model path:

  Startpoint: IN (input port clocked by CLK)
  Endpoint: ff1 (rising edge-triggered flip-flop clocked by CLK')
  Path Group: CLK
  Path Type: max
  Scenario: model_validation

  Point                        Fanout    Cap      Trans       Incr       Path
-------------------------------------------------------------------------------
```

```
  clock CLK (rise edge)                                          0.00       0.00
  clock network delay (propagated)                               0.00       0.00
  input external delay                                           0.20       0.20 f
  IN (in)                                        0.00            0.00       0.20 f
  IN (net)                        1     1.00
  U1/A (ND2)                                     0.00            0.00       0.20 f
  U1/Z (ND2)                                     0.17            0.67 &     0.87 r
  n1 (net)                        1     1.23
  ff1/D (FD1)                                    0.17            0.00 &     0.87 r
  data arrival time                                                         0.87

  clock CLK' (rise edge)                                         0.60       0.60
  clock source latency                                           0.00       0.60
  CLK (in)                                       0.00            0.00       0.60 f
  CLK (net)                       3     3.00
  U2/A (IV)                                      0.00            0.00       0.60 f
  U2/Z (IV)                                      0.14            0.52       1.13 r
  n2 (net)                        1     1.00
  ff1/CP (FD1)                                   0.14            0.00       1.13 r
  library setup time                                            -0.80       0.33
  data required time                                                        0.33
--------------------------------------------------------------------------------
  data required time                                                        0.33
  data arrival time                                                        -0.87
--------------------------------------------------------------------------------
  slack (VIOLATED)                                                         -0.54


The dominant exceptions are:
         None


Stage 0 with difference :
  Point                        Fanout    Cap      Trans       Incr       Path
  --------------------------------------------------------------------------------
  U1/Z (ND2)                                      0.17       0.67 &     0.87 r
  U1/Z (ND2)                                      0.17       0.67 &     0.87 r
  ff1/D (FD1)                                     0.18       0.01 &     0.88 r
  ff1/D (FD1)                                     0.17       0.00 &     0.87 r
  --------------------------------------------------------------------------------
  Path    Aggressor                   Screen                   Switching Bump
          Net                                                 (ratio of VDD)
  -----  -----------------------     -----------------          ---------------
   0     n5                          active                     0.03
   1     n5                          screened_by_macro_model
                                                                0.00
--------------------------------------------------------------------------------
```

SEE ALSO

```
create_ilm(2)
extract_model(2)
hier_modeling_version(3)
```

multi_core_allow_overthreading

Controls overthreading with more than one thread per CPU core.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Given a maximum limit on the CPU cores that the current PrimeTime process can use, the count of simultaneously active threads can exceed that limit. This is determined differently for different threaded algorithms and commands. The degree of overthreading is set up with the guarantee that most of the time the cores usage limit that you set is honored. However, it is possible that for very short intervals, the usage might exceed that limit.

If this is absolutely undesirable, set the **multi_core_allow_overthreading** variable to **false** to guarantee that the process cores utilization never exceeds the user maximum cores limit. This would result in reduced multicore analysis performance.

Note that modifying this variable only affects algorithms invoked within Tcl commands launched after you set the variable.

SEE ALSO

`set_host_options(2)`

multi_scenario_enable_analysis

Enables Distributed Multi-Scenario Analysis (DMSA) mode.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable the Distributed Multi-Scenario Analysis (DMSA) flow in the PrimeTime tool.

Verifying a chip design generally requires multiple PrimeTime runs to check correct operation under different operating conditions and different operating modes. A specific combination of operating conditions and operating modes for a given chip design is called a scenario.

The PrimeTime tool can analyze several scenarios in parallel using DMSA. Instead of analyzing each scenario in sequence, DMSA uses a master PrimeTime process that sets up and controls multiple worker processes to execute the timing analysis for each scenario. The processing of scenarios can be distributed onto different hosts running in parallel, thus reducing the overall turnaround time.

After you enable DMSA mode, you cannot disable it for the rest of the PrimeTime session.

The variable can only be set to **true** if these conditions are met:

- No design has been loaded into PrimeTime
- No other distributed mode has been activated
- The GUI has not been started

Changing the **multi_scenario_enable_analysis** variable to **true** also sets the variable **pt_shell_mode** to **primetime_master**.

SEE ALSO

pt_shell_mode(3)
create_scenario(2)
current_session(2)
start_hosts(2)

multi_scenario_fault_handling

Controls how fatal errors are handled in remote processes.

TYPE

string

DEFAULT

ignore

DESCRIPTION

Controls how fatal errors are handled in remote processes. The following values are allowed:

- **exit** - If the master detects fatal errors in the remote processes, it shuts down the entire analysis after all the executing tasks have been processed. Before exiting, the master reports which scenarios have caused a fatal error.
- **ignore** (default) - If the master detects fatal errors in the remote processes, it removes the scenarios that caused a fatal error from the current session when all the executing tasks have been processed. It then reruns the command encountering the fatal error on the remaining scenarios in command focus. It repeats this process until the command either succeeds or the command focus is depleted of all scenarios. If there are no scenarios left in command focus, the command does not run again. If all the scenarios in the session are removed, the session is removed. If the resources required to sustain a session are no longer available, the current session is removed. For the resource requirements needed to sustain a session, see the **current_session** man page.

SEE ALSO

```
current_scenario(2)
current_session(2)
```

multi_scenario_license_mode

Specifies the DMSA licensing mode, either scenario-based or core-based.

TYPE

string

DEFAULT

scenario

DESCRIPTION

Set this variable at the DMSA master to either "**scenario**" (the default) or "**core**" to specify either scenario-based or core-based licensing:

- In scenario-based licensing, one license is required for each running scenario, without considering the number of cores being used.
- In core-based licensing, one license is required for running each 32 cores, without considering the number of scenarios being run.

Here are the specific actions carried out in each licensing mode:

- **scenario** (default) - For every license feature required by the design in a scenario, a license key for each feature is checked out from the master by each concurrently running scenario in order to perform the analysis.
- **core** - When you set the variable to **core**, the master attempts to check out one PrimeTime-ADV feature license. If the license cannot be checked out, the variable is automatically set back to **scenario**. If the license can be checked out, the variable remains set to **core** and the master allows up to 32 cores of concurrent worker execution without additional licenses. For example, 4 workers configured using set_host_options -max_cores 8 can all use PrimeTime-ADV licensed capabilities.

For every additional 32 cores of concurrent execution (or part thereof), an additional PrimeTime-ADV feature license is checked out by the master. For example, 5 workers configured using set_host_options -max_cores 8 would require the master to check out a total of two PrimeTime-ADV feature licenses. If the master cannot acquire additional PrimeTime-ADV feature licenses, the workers requiring the additional licenses are queued for later execution.

After the variable is set to **core**, one PrimeTime-ADV license must remain checked out at the master at all times. Attempting to remove all PrimeTime-ADV feature licenses removes all but one license key.

Note: Only the following feature licenses are controlled by the **core** setting (PrimeTime, PrimeTime-SI, PrimeTime-ADV, PrimeTime-ADV-PLUS, PrimePower). All other feature licenses behave as described under the **scenario** setting.

SEE ALSO

current_scenario(2)
current_session(2)
set_host_options(2)

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

integer

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** variable is 100.

SEE ALSO

`multi_scenario_merged_error_log(3)`

multi_scenario_merged_error_log

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

TYPE

string

DEFAULT

none

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. All errors, warnings, and informational messages are displayed as in regular PrimeTime, with the addition of the name of the scenario where the message occurred. 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 at the master (slave informational messages are not displayed at the master). At each slave, the errors and warnings are written in full to the slave log file.

SEE ALSO

`multi_scenario_merged_error_limit(3)`

multi_scenario_message_verbosity_level

Specifies 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.

You can set this variable to one of these values:

- **low** - The master prints messages for errors, fatal errors, and task failures.
- **default** - The master prints messages for licensing, errors, warnings, fatal errors, task status, and task failures.

SEE ALSO

`multi_scenario_merged_error_log(3)`

multi_scenario_working_directory

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

TYPE

string

DEFAULT

(current working directory)

DESCRIPTION

In multi-scenario analysis, the working directory can be explicitly specified using the **multi_scenario_working_directory** variable. This defines the root working directory for all multi-scenario analysis data, including log files. The working directory must be visible to the master and all the slaves. You must have write permissions in the specified directory. If no value is specified for the **multi_scenario_working_directory** variable, the working directory defaults to the directory from which the analysis was invoked.

SEE ALSO

`multi_scenario_working_directory(3)`

mv_allow_pg_pin_reconnection

Enables reconnection of a supply net to a PG pin.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable reconnection of a supply net to a PG pin that already has a connection.

This variable allows reconnections on only PG pins. Reconnection of a supply net to a port is not allowed even with the variable is set to **true**.

SEE ALSO

`connect_supply_net(2)`

mv_input_enforce_simple_names

Enforces usage of simple names for restricted commands as per the IEEE 1801 Unified Power Format (UPF) standard.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether hierarchical names are accepted when you specify an argument that requires simple names, according to the UPF standard.

By default, the tool accepts hierarchical names even if the UPF standard requires them to be simple.

If you set this variable to **true**, the restricted command causes the tool to error out when you use hierarchical names. Note that this variable does not affect the **create_supply_net** and **create_supply_set** commands, which always require simple names.

SEE ALSO

```
connect_supply_net(2)
create_power_domain(2)
create_power_switch(2)
create_supply_port(2)
```

mv_upf_allow_negative_voltage

Causes the tool to check for negative power and ground supply voltages set for logic pins of macros and interface timing models, and disables timing analysis for these pins.

TYPE

boolean

DEFAULT

false

DESCRIPTION

You can set this variable to the following values:

- **false** (the default) -- The tool does not perform any check for negative power or ground supply voltages associated with logic pins.
- **true** -- The tool checks for negative power and ground supply voltages for the logic pins of macros and interface timing model cells. A negative supply voltage can be specified in the library cell definition or UPF supply net. If it detects a negative voltage, it applies the **set_disable_timing** command to that pin, which disables all timing paths through the pin. The **set_disable_timing** command is written by the **write_script** and **write_sdc** commands, and appears in the HyperScale model for the block.

SEE ALSO

`report_disable_timing(2)`

```
    set_disable_timing(2)
    set_voltage(3)
```

mw_design_library

Specifies the name of the design library for the **read_milkyway** command.

TYPE

string

DEFAULT

"" (empty)

DESCRIPTION

This variable specifies the name of the design library for the **read_milkyway** command. If you run the **read_milkyway** command without specifying the design library name, the command reads this variable to get the library name. If you run the **read_milkyway** command with the design library name, this variable is set to that name.

SEE ALSO

`read_milkyway(2)`

mw_logic0_net

Specifies the name of constant zero net for the **read_milkyway** command.

TYPE

string

DEFAULT

VSS

DESCRIPTION

This variable specifies the name of the net is the logic low net for the **read_milkyway** command. When the **read_milkyway** command is reading a design and sees a net by this name, the command treats the net as if it were tied to logic 0.

SEE ALSO

```
read_milkyway(2)
mw_logic1_net(3)
```

mw_logic1_net

Specifies the name of the constant one net for the **read_milkyway** command.

TYPE

string

DEFAULT

VDD

DESCRIPTION

This variable specifies the name of the net is the logic high net for the **read_milkyway** command. When the **read_milkyway** command is reading a design and sees a net by this name, the command treats the net as if it were tied to logic 1.

SEE ALSO

```
read_milkyway(2)
mw_logic0_net(3)
```

net_attributes

Describes the predefined application attributes for net objects.

DESCRIPTION

activity_source

Type: string
Returns the source of switching activity information for the net: file, set_switching_activity, set_case_analysis, propagated, implied, default, or UNINITIALIZED.

aggressors

Type: string
Returns the aggressor nets that affect the victim net.

annotated_delay_delta_max

Type: float
Returns the maximum of the annotated_delay_delta_max attributes on all net arcs.

annotated_delay_delta_min

Type: float
Returns the minimum of the annotated_delay_delta_min attributes on all net arcs.

annotated_transition_delta_max

Type: float
Returns the maximum of the annotated_transition_delta_max attributes on all leaf pins and ports.

annotated_transition_delta_min

Type: float
Returns the minimum of the annotated_transition_delta_min attributes on all leaf pins and ports.

area

Type: float
Returns the estimated area of the net, calculated using a wire load model.

ba_capacitance_max

Type: float
Returns the back-annotated capacitance on the net for maximum conditions, set with the set_load or read_parasitics command.

ba_capacitance_min

Type: float
Returns the back-annotated capacitance on the net for minimum conditions, set with the set_load or read_parasitics command.

ba_resistance_max

Type: float
Returns the back-annotated resistance on the net for maximum conditions, set with the set_resistance or read_parasitics command.

ba_resistance_min

Type: float
Returns the back-annotated resistance of the net for minimum conditions, set with the set_resistance or read_parasitics command.

base_name

Type: string
Returns the leaf name of the net. For example, the base name of net i1/i1z1 is i1z1. You cannot modify this attribute.

coupling_capacitors

Type: string
Lists the cross-coupling capacitance values in main library units. With this attribute the nets are explicitly identified. For example,

```
{u1a/A (n1) u2b/Z (n2) 0.40 not_filtered}
{n1:3 (n1) n2:5 (n2) 0.03 filtered_by_accum_noise_peak}
{in_port (n1) n3:7 (n3) 0.01 filtered_by_accum_noise_peak}
```

dont_touch

Type: boolean
Returns true if the net is protected from change by the set_dont_touch command. When this attribute is set to true, buffers cannot be inserted on the net by ECO fixing commands (fix_eco_drc, fix_eco_power, and fix_eco_timing). By default, driver and load cells connected to the net can still be modified or removed by ECO fixing commands. This attribute exists for a net only after the attribute is created by the set_dont_touch command.

early_fall_clk_net_delta_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the dynamic (crosstalk) component of net delay for early falling clock transitions on the net.

early_fall_clk_net_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the static component of net delay for early falling clock transitions on the net.

early_fall_data_net_delta_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the dynamic (crosstalk) component of net delay for early falling data transitions on the net.

early_fall_data_net_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the static component of net delay for early falling data transitions on the net.

early_rise_clk_net_delta_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the dynamic (crosstalk) component of net delay for early rising clock transitions on the net.

early_rise_clk_net_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the static component of net delay for early rising clock transitions on the net.

early_rise_data_net_delta_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the dynamic (crosstalk) component of net delay for early rising data transitions on the net.

early_rise_data_net_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the static component of net delay for early rising data transitions on the net.

effective_aggressors

Type: string
Lists the cross-coupled aggressor nets that have an effect large enough to be analyzed on the net as a victim. Other cross-coupled aggressor nets are excluded from the list. For more information, see the description of the si_xtalk_bumps attribute.

effective_coupling_capacitors

Type: string
Lists the effective cross-coupling capacitance values in main library units. Only the capacitors that are

not excluded are shown.

escaped_full_name

Type: string
Returns the full hierarchical name of the net with any literal hierarchy characters escaped with a backslash.

full_name

Type: string
Returns the full hierarchical name of the net. For example, the full name of net i1z1 within cell i1 is i1/i1z1. The full_name attribute is not affected by the current instance setting. The full_name attribute is read-only.

glitch_count

Type: float
Returns the number of glitch transitions on the net for the duration of power_simulation_time (a design attribute).

glitch_rate

Type: double
Returns the rate at which glitch transitions occur on the net, equal to glitch_count/power_simulation_time

has_detailed_parasitics

Type: boolean
Returns true if any part of the net has annotated detailed parasitics, even if only one segment of the net at a different level of hierarchy.

has_valid_parasitics

Type: boolean
Returns true if the net has an annotated pi model, or if all segments of the net are annotated with properly connected detailed parasitics that form a valid representation of physical interconnection between drivers and loads.

is_clock_network

Type: boolean
Returns true if the net is in the combinational fanout of a clock source, that is, the is_clock_network attribute is true on any of the leaf pins or ports of the net.

is_clock_source_network

Type: boolean
Returns true if the net is part of clock source latency network, that is, the is_clock_source_network attribute is true on any of its leaf pins or ports.

is_design_mismatch

Type: boolean
Returns true if the net has a mismatch between the block and the top-level design.

is_ideal

Type: boolean
Returns true if the net has been marked ideal using the set_ideal_network command.

is_power_control_signal_net

Type: boolean
Returns true if the signal net is used to control any power rail or PrimePower rail mapping modes.

late_fall_clk_net_delta_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the dynamic (crosstalk) component of net delay for late falling clock transitions on the net.

late_fall_clk_net_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the static component of net delay for late falling clock transitions on the net.

late_fall_data_net_delta_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the dynamic (crosstalk) component of net delay for late falling data transitions on the net.

late_fall_data_net_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the static component of net delay for late falling data transitions on the net.

late_rise_clk_net_delta_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the dynamic (crosstalk) component of net delay for late rising clock transitions on the net.

late_rise_clk_net_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the static component of net delay for late rising clock transitions on the net.

late_rise_data_net_delta_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the dynamic

(crosstalk) component of net delay for late rising data transitions on the net.

late_rise_data_net_derate_factor

Type: float
Returns the derating factor specified by the set_timing_derate command that applies to the static component of net delay for late rising data transitions on the net.

leaf_drivers

Type: collection
Returns a collection of leaf driver pins and ports of the net.

leaf_loads

Type: collection
Returns a collection of leaf load pins and ports of the net.

max_slack

Type: float
Returns the minimum of the max_slack of all leaf pins and ports. This attribute is computed only when the timing_save_pin_arrival_and_slack variable is set to true.

min_slack

Type: float
Returns the minimum of the min_slack attribute on all leaf pins and ports. This attribute is computed only when the timing_save_pin_arrival_and_slack variable is set to true.

net_resistance_max

Type: float
Returns the resistance of the net for maximum conditions. The value be computed from wire load models, set by the set_resistance command, or annotated by the read_parasitics command.

net_resistance_min

Type: float
Returns the resistance of the net for minimum conditions. The value be computed from wire load models, set by the set_resistance command, or annotated by the read_parasitics command.

effective_aggressors

Type: string
Lists the cross-coupled aggressor nets that have an effect large enough to be analyzed on the net as a victim. Other cross-coupled aggressor nets are excluded from the list. For more information, see the description of the si_xtalk_bumps attribute.

effective_coupling_capacitors

Type: string
Lists the effective cross-coupling capacitance values in main library units. Only the capacitors that are

not excluded are shown.

number_of_aggressors

Type: integer
Returns the total number of cross-coupled aggressor nets, including both effective and ignored aggressor nets.

number_of_coupling_capacitors

Type: integer
Returns the total number of coupling capacitors, including both effective and ignored capacitors.

number_of_effective_aggressors

Type: integer
Returns the number of cross-coupled aggressor nets that have an effect large enough to be analyzed on the net as a victim.

number_of_effective_coupling_capacitors

Type: integer
Returns the number of coupling capacitors on the net that have an effect large enough to be analyzed.

number_of_leaf_drivers

Type: integer
Returns the number of driver leaf pins that are connected to the net. Because a leaf pin is connected to a leaf cell, this attribute does not include pins at hierarchical boundaries. For hierarchical nets, this attribute reflects the total number of driver pins connected to all net segments.

number_of_leaf_loads

Type: integer
Returns the number of load leaf pins that are connected to the net. Because a leaf pin is connected to a leaf cell, this attribute does not include pins at hierarchical boundaries. For hierarchical nets, this attribute reflects the total number of load pins connected to all net segments.

object_class

Type: string
Returns the class of the object, which is "net". You cannot set this attribute.

parent_cell

Type: collection
Returns the parent hierarchical cell containing the net.

pin_capacitance_max

Type: float
Returns the sum of all pin capacitance values of the net for maximum conditions. You cannot modify this attribute.

pin_capacitance_max_fall

Type: float
Returns the maximum fall capacitance of all pins and ports for the net. You cannot modify this attribute.

pin_capacitance_max_rise

Type: float
Returns the maximum rise capacitance of all pins and ports for the net. You cannot modify this attribute.

pin_capacitance_min

Type: float
Returns the sum of all pin capacitance values of the net for minimum conditions. You cannot modify this attribute.

pin_capacitance_min_fall

Type: float
Returns the minimum fall capacitance of all pins and ports for the net. You cannot modify this attribute.

pin_capacitance_min_rise

Type: float
Returns the minimum rise capacitance of all pins and ports for the net. You cannot modify this attribute.

power_base_clock

Type: string
Returns the name of the base clock associated with this net. If the net belongs to multiple clock domains, the attribute is set to the fastest of the clocks.

rc_annotated_segment

Type: boolean
Returns true if the specific net segment has annotated parasitics. For two segments at different levels of hierarchy (for example, n1 and h1/n1), the attribute values can differ.

rc_network

Type: string
Returns a string describing the parasitic data that has been back-annotated on the net, including resistor values (in KOhms) and capacitor values (in pF).

rc_network_with_sensitivity

Type: string
Returns the full RC network of the net with sensitivity values included. This attribute is read-only.

relative_toggle_rate

Type: double

Returns the number of toggles per period of the power_base_clock. If the design does not have a defined clock, this attribute is undefined.

si_double_switching_slack

Type: float
Returns the slack value if the net has double-switching slack on the victim net.

si_has_double_switching

Type: boolean
Returns true if the net has a double-switching violation. Double-switching analysis needs to be enabled.

si_xtalk_bumps

Type: string
Lists each aggressor net and the voltage bumps that rising and falling aggressor transitions induce on the victim net (worst of rising minimum or maximum bumps and worst of falling minimum or maximum bumps, each expressed as a decimal fraction of the rail-to-rail voltage), or gives the reason that an aggressor net has no effect on the victim net.

si_xtalk_bumps_max_fall

Type: string
Lists each aggressor net and the voltage induced on the victim net (expressed as a decimal fraction of the rail-to-rail voltage) for a maximum fall transition.

si_xtalk_bumps_max_rise

Type: string
Lists each aggressor net and the voltage induced on the victim net (expressed as a decimal fraction of the rail-to-rail voltage) for a maximum rise transition.

si_xtalk_bumps_min_fall

Type: string
Lists each aggressor net and the voltage induced on the victim net (expressed as a decimal fraction of the rail-to-rail voltage) for a minimum fall transition.

si_xtalk_bumps_min_rise

Type: string
Lists each aggressor net and the voltage induced on the victim net (expressed as a decimal fraction of the rail-to-rail voltage) for a minimum rise transition.

si_xtalk_composite_aggr_max_fall

Type: collection
Returns a collection of aggressors in a potential composite aggressor group for a maximum fall transition.

si_xtalk_composite_aggr_max_rise

Type: collection
Returns a collection of aggressors in a potential composite aggressor group for a maximum rise transition.

si_xtalk_composite_aggr_min_fall

Type: collection
Returns a collection of aggressors in a potential composite aggressor group for a minimum fall transition.

si_xtalk_composite_aggr_min_rise

Type: collection
Returns a collection of aggressors in a potential composite aggressor group for a minimum rise transition.

si_xtalk_used_ccs_max_fall

Type: boolean
Returns true if the net was analyzed for crosstalk delay using CCS timing models for maximum fall timing constraints and transitions.

si_xtalk_used_ccs_max_rise

Type: boolean
Returns true if the net was analyzed for crosstalk delay using CCS timing models for maximum rise timing constraints and transitions.

si_xtalk_used_ccs_min_fall

Type: boolean
Returns true if the net was analyzed for crosstalk delay using CCS timing models for minimum fall timing constraints and transitions.

si_xtalk_used_ccs_min_rise

Type: boolean
Returns true if the net was analyzed for crosstalk delay using CCS timing models for minimum rise timing constraints and transitions.

static_probability

Type: float
Returns the static probability of the net, which is the probability that the net has the logic value 1.

switching_power

Type: double
Returns the switching power of the net in watts, which is the power dissipated by the charging and discharging of the capacitance of the net.

toggle_count

Type: float
Returns the number of transitions of the net during the duration of the power_simulation_time attribute (a design attribute).

toggle_rate

Type: double
Returns the rate at which transitions occur on the net. It is equal to toggle_count/power_simulation_time

total_capacitance_max

Type: float
Returns the sum of all pin capacitance values and the wire capacitance of the net for maximum conditions. This attribute is read-only.

total_capacitance_min

Type: float
Returns the sum of all pin capacitance values and the wire capacitance of the net for minimum conditions. This attribute is read-only.

total_ccs_capacitance_max_fall

Type: float
Returns the total capacitance, including both wire capacitance and maximum of falling CCS receiver capacitance. For example, total_ccs_capacitance_max_fall is the sum of wire capacitance and maximum of fall_c1/fall_c2.

total_ccs_capacitance_max_rise

Type: float
Returns the total capacitance, including both wire capacitance and maximum of rising CCS receiver capacitance. For example, total_ccs_capacitance_max_rise is the sum of wire capacitance and maximum of rise_c1/rise_c2.

total_ccs_capacitance_min_fall

Type: float
Returns the total capacitance, including both wire capacitance and minimum of falling CCS receiver capacitance. For example, total_ccs_capacitance_min_fall is the sum of wire capacitance and minimum of fall_c1/fall_c2.

total_ccs_capacitance_min_rise

Type: float
Returns the total capacitance, including both wire capacitance and minimum of rising CCS receiver capacitance. For example, total_ccs_capacitance_min_rise is the sum of wire capacitance and minimum of rise_c1/rise_c2.

total_coupling_capacitance

Type: float

Returns the total cross-coupling capacitance (in main library units) of the net as a victim.

total_effective_coupling_capacitance

Type: float
Returns the total effective cross capacitance (in main library units) of the net as a victim, including only the capacitors that have a large enough effect to be considered for analysis.

user_global_coupling_separated

Type: boolean
Returns true if this net has been globally separated by the set_coupling_separation command.

user_pairwise_coupling_separated

Type: collection
Returns a collection of other nets that have been pairwise-separated from the net by the set_coupling_separation command.

wire_capacitance_max

Type: float
Returns the wire capacitance of the net for maximum conditions. The value can be computed from wire load models, set by the set_load command, or annotated by the read_parasitics command.

wire_capacitance_min

Type: float
Returns the wire capacitance of the net for minimum conditions. The value can be computed from wire load models, set by the set_load command, or annotated by the read_parasitics command.

x_coordinate_max

Type: float
Returns the maximum x-coordinate of the area occupied by the net.

x_coordinate_min

Type: float
Returns the minimum x-coordinate of the area occupied by the net.

y_coordinate_max

Type: float
Returns the maximum y-coordinate of the area occupied by the net.

y_coordinate_min

Type: float
Returns the minimum y-coordinate of the area occupied by the net.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

non_auto_fixable_violations

This man page describes **non_auto_fixable** hierarchical boundary check violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

The following violations are **non_auto_fixable**:

```
boundary_ideal_network
clock_attributes
clock_mapping
clock_relations
clock_uncertainty
env_variables
global_timing_derate
library_mapping
operating_conditions
```

The **non_auto_fixable** violations invalidate the HyperScale flow. To use HyperScale, you must fix **non_auto_fixable** violations in the beginning or middle of the design flow.

WHAT NEXT

To resolve **non_auto_fixable** violations,

- Focus on aligning block-level constraints to top-level constraints, such as environment and clocks.
- If necessary, improve the specification of top-level constraints for compatibility with hierarchical analysis.

EXAMPLES

The following verbose report shows **non_auto_fixable** HyperScale boundary violations.

```
pt_shell> report_constraint -all_violators -verbose -boundary_check_include {non_auto_fixable}
```

```
****************************************
Report : constraint
        -all_violators
        -verbose
Design : top
 ...
****************************************

HyperScale constraints report

   Constraint: clock_attributes

   Instance      Attribute name    Block level (clock)     Top level (clock)
   -------------------------------------------------------------------------------
   core2         is_propagated     false (CLK)             true (top_CLK)
   core3         sources           {core3/clk1} (CLK)      {core3/clk2} (top_CLK)

   Constraint: clock_relations

   Instance      Clock relation    Block level             Top level
   -------------------------------------------------------------------------------
   core2         async             {int_CLK} <-> {top_CLK} {core2/int_CLK} <-> {}
   core2         async             {vCLK} <-> {top_CLK}    {top_vCLK} <-> {}
```

SEE ALSO

report_constraint(2)
auto_fixable_violations(3)
boundary_ideal_network_violations(3)
clock_attributes_violations(3)
clock_mapping_violations(3)
clock_relations_violations(3)
clock_uncertainty_violations(3)
env_variables_violations(3)
global_timing_derate_violations(3)
library_mapping_violations(3)
operating_conditions_violations(3)

old_port_voltage_assignment

Reverts to a port's previous voltage assignment.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the tool infers a port voltage from its load or driver pins. If you set this variable is set to **true**, the tool reverts to the previous voltage assignment mode for ports and no longer infers port voltages from load or driver pins.

operating_conditions_violations

This man page describes **operating_conditions** hierarchical boundary check violations shown by the **report_constraint** command in the HyperScale flow.

DESCRIPTION

An **operating_conditions** violation indicates differences in the operating condition related settings.

Timing constraint commands referring to libraries include (and not limited to) **set_operating_condition**, **set_temperature set_voltage**.

Note: Do not save and check the operating condition settings applied to specific design objects such as pins, cells. This check only covers design-wide operating conditions, or settings on block boundary ports.

WHAT NEXT

This is a **non_auto_fixable** violation. You must resolve the difference in operating condition settings. Consistent usage and settings of operating condition is required to ensure consistent results from the timing analysis. The difference can be resolved by the commands listed previously. If the violation is a side effect of library mismatch, resolving library_mapping violation is required.

EXAMPLES

The following verbose report shows mismatches in the operating condition between the top and block.

```
pt_shell> report_constraint -boundary_check_include {operating_conditions} \
          -all_violators -verbose

****************************************
Report : constraint
        -verbose
 ...
****************************************

HyperScale constraints report
```

```
Constraint: operating_conditions

Instance   Setting           Block         Top
-------------------------------------------------------------
blk        Max Library       /my/libraries/pt_lib.db:pt_lib
                                           /remote/libraries/pt_lib_max.db:pt_lib_max
blk        Max Condition     WCCOM         nom_pvt
blk        Analysis Type     bc_wc         Single
blk        Min Temperature   125           70
```

SEE ALSO

report_constraint(2)
set_operating_conditions(2)
set_temperature(2)
set_voltage(2)
non_auto_fixable_violations(3)

parasitic_corner_name

Specifies the parasitic corner to be loaded by the **read_parasitics** command.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable specifies the parasitic corner to be loaded by the **read_parasitics** command. Setting this variable is optional. The tool behavior in relation to parasitic corners is as follows:

- If you set the variable to a specified corner:
    1. If the tool reads SPEF or SBPF files with multiple corners, the **read_parasitics** command annotates the nets with only the data of the specified corner.
    2. If the specified corner does not exist in the SPEF or SBPF files, the tool issues an error.
- If the variable is not set:
    1. If the tool reads SPEF or SBPF files with multiple corners, the tool issues an error.
    2. Otherwise, the tool reads the parasitic data.
- You can set this variable at any time before reading the parasitic data.
- After reading parasitic data, you cannot create a new or destroy an existing parasitic corner. This means that you can set the variable only to an existing parasitic corner.

SEE ALSO

read_parasitics(2)

parasitic_explorer_enable_analysis

Enables or disables Parasitic Explorer features, which provides parasitic analysis features on GPD parasitics.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, this variable is set to **false**, which disables Parasitic Explorer.

To enable Parasitic Explorer, set this variable to **true**, so that the **read_parasitics** makes use of multicorner GPD parasitcis and various parasitics properties. Additionally, this variable enables various GPD query/reporting commands.

If you set this variable to **true**, you must also do the following:

1. Obtain a PrimeTime ADV+ license. You cannot use Parasitic Explorer without a license.
2. Use the **read_parasitics -format_gpd** command to read in the GPD parasitics for your design.

SEE ALSO

```
read_parasitics(2)
get_ground_capacitors(2)
get_coupling_capacitors(2)
```

```
get_resistors(2)
report_gpd_properties(2)
report_gpd_config(2)
```

parasitics_cap_warning_threshold

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

TYPE

float

DEFAULT

0.0

DESCRIPTION

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

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

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

SEE ALSO

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

parasitics_log_file

Specifies the location of the output of parasitic commands when run in a background process.

TYPE

string

DEFAULT

parasitics_command.log

DESCRIPTION

The tool might launch a side process to perform parasitic operations in parallel with portions of the main run. This variable determines where the output data of the side process should go. By default, it goes to a file named parasitics_command.log in the current working directory. To modify the file name, set this variable.

The **parasitics_log_file** variable works only when multiple CPU cores are used. If only one CPU core is specified or available, this variable setting has no effect and the tool writes out parasitic information to the default log file.

If the **parasitics_log_file** variable is set to empty string "", then the output parasitic info is displayed in the default log file.

SEE ALSO

`read_parasitics(2)`

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

integer

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 runtimes caused by large numbers of nodes in an annotated parasitic network. If the **read_parasitics** command finds a number of nodes that exceeds the value of the **parasitics_warning_net_size** variable (default 10000), the PARA-003 message warns you that extended runtime could occur. If the **read_parasitics** command finds a number of nodes that exceeds the value of the **parasitics_rejection_net_size** variable (default 20000), the network is rejected and automatically replaced by a lumped capacitance, and you receive a PARA-004 message warning.

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

SEE ALSO

read_parasitics(2)
parasitics_warning_net_size(3)

parasitics_res_warning_threshold

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

TYPE

float

DEFAULT

0.0

DESCRIPTION

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

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

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

SEE ALSO

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

parasitics_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 runtime.

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 runtime. 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 runtimes caused by large numbers of nodes in an annotated parasitic network. If the **read_parasitics** command finds a number of nodes that exceeds the value of the **parasitics_warning_net_size** variable (default 10000), you receive a PARA-003 message warning you that extended runtime could occur. If the **read_parasitics** command finds a number of nodes that exceeds the value of the **parasitics_rejection_net_size** variable (default 20000), the network is rejected and automatically replaced by a lumped capacitance. You receive a PARA-004 message warning you of that action.

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

SEE ALSO

```
read_parasitics(2)
parasitics_rejection_net_size(3)
```

path_group_attributes

Describes the predefined application attributes for path_group objects.

DESCRIPTION

full_name

Type: string
Returns the name of the path group. Path groups are created by group_path or implicitly using create_clock. This attribute is read-only; you cannot change the setting.

object_class

Type: string
Returns the class of the object, which is a constant equal to path_group. You cannot set this attribute.

weight

Type: float
Returns the cost function weight assigned to this path group. The weight of a group specifies how much the group influences the total maximum delay cost and can be used to guide optimization. You can specify the weight with group_path.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

pba_all_paths_endpoint_time_limit

Defines a time limit (milliseconds) for "all paths" mode exhaustive path-based analysis.

TYPE

string (Range: 1 to large number, or string "infinity")

DEFAULT

infinity

DESCRIPTION

This variable applies to "all paths" mode exhaustive path-based analysis performed by the following commands:

```
report_timing -pba_mode exhaustive ...
get_timing_paths -pba_mode exhaustive ...
report_constraint -pba_mode exhaustive ...
report_global_timing -pba_mode exhaustive ...
report_qor -pba_mode exhaustive ...
```

This variable is only applicable to the "all paths" mode of exhaustive path-based analysis. "all paths" mode needs to be activated by setting "pba_exhaustive_endpoint_path_limit" variable to infinity.

"all paths" algorithm performs exhaustive path-based analysis on a set of endpoints. The time spent analyzing an endpoint depends on the topological complexity of the paths ending at that endpoint.

This variable is a means for the user to set the maximum time "all paths" algorithm may spent analyzing an endpoint. The time unit of this variable is milliseconds.

If this variable is set, then each endpoint will be subjected to a time limit. If time limit is exceeded for any endpoint, then that endpoint analysis will halt with a UITE-580 warning message being issued.

All the endpoints for which UITE-580 was issued because of exceeded time limit will be added to the

pba_exhaustive_endpoint_limit_exceeded_pins design attribute.

Due to the precence of heavily threaded algorithms it is not possible to guarantee determinism from a run to another. The pba_exhaustive_endpoint_limit_exceeded_pins design attribute together with the content and count of UITE-580 messages may differ from run to run.

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
pba_exhaustive_endpoint_path_limit(3)
```

pba_derate_only_mode

Specifies that only the path derates are reevaluated during path-based analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable applies to the path-based analysis performed during the **get_timing_paths** and **report_timing** commands when you specify the **-pba_mode** option. This variable controls whether regular path-based analysis (path-specific slew propagation) effects are performed during a path-based analysis in addition to adjusting the deratings on the path according to the path-based conditions.

If the variable is set to **false** (the default), the tool performs both regular path-based analysis and derating adjustment during a path-based analysis. This option removes the maximum amount of pessimism from the design, but the runtime can be long.

If you set the variable to **true**, the tool only adjusts the derating according to the path-based conditions during the path-based analysis.

This variable is useful in the advanced OCV flow, parametric OCV flow, and simultaneous multivoltage analysis (SMVA) used with a derating method and is recommended in these flows to achieve the fastest runtime.

Note that advanced or parametric OCV path-based analysis is applied only if user-specified advanced or parametric OCV information exists.

Also note that in parametric OCV analysis, the random variation pessimism of graph-based analysis over path-based analysis is zero. Therefore, running path-based analysis with this variable set to **true** in

parametric OCV only removes pessimism resulting from systematic variation, that is, distance-based derating.

SEE ALSO

```
get_timing_paths(2)
read_aocvm(2)
report_timing(2)
```

pba_enable_path_based_physical_exclusivity

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

TYPE

integer

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), each delay calculation stage is evaluated independently of the other stages in the path. For instance, consider two clocks that 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.

If you set this variable to **true**, the path is recalculated multiple times to consider each possible victim and aggressor combination across the physically-exclusive clocks. In this case, aggressors from CLK1 and CLK2 cannot simultaneously attack different stages across the path. This removes the pessimism of the stage-based approach, but at the cost of additional runtime.

It is recommended to use the default setting of **false** for most analysis, but set the variable to **true** for the final signoff run if additional pessimism removal is needed during path-based analysis.

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
```

pba_enable_xtalk_delay_ocv_pessimism_reduct

Reduces the pessimism during path-based crosstalk delay analysis due to clock on-chip variation (OCV).

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether PrimeTime SI reduces the pessimism due to clock on-chip variation (OCV) on the victim and aggressor arrivals during path-based crosstalk delay analysis. You can set this variable to one of these values:

- **false** (the default) - Does not reduce the pessimism due to clock on-chip-variation (OCV) on the victim and aggressor arrivals.
- **true** - Reduces the pessimism due to clock on-chip-variation (OCV) on the victim and aggressor arrivals. This pessimism reduction is computationally intensive and should be used only when the clock path is long, and the on-chip variation is large.

To use this feature, you must enable clock reconvergence pessimism removal (CRPR) by setting the **timing_remove_clock_reconvergence_pessimism** variable to **true**.

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
timing_remove_clock_reconvergence_pessimism(3)
```

pba_exhaustive_endpoint_path_limit

Defines a limit for exhaustive path-based analysis.

TYPE

string (Range: 1 to large number, or string "infinity")

DEFAULT

infinity

DESCRIPTION

This variable applies to the exhaustive path-based analysis performed by the following commands:

```
report_timing -pba_mode exhaustive ...
get_timing_paths -pba_mode exhaustive ...
report_constraint -pba_mode exhaustive ...
report_global_timing -pba_mode exhaustive ...
report_qor -pba_mode exhaustive ...
```

Exhaustive analysis is computationally intensive, and it is intended to be used only when the design is approaching signoff.

In certain topologically complex designs, you get full path coverage with faster runtime when you leave the variable set to the default, the string "infinity". This selects the "all paths" mode.

Alternatively, for less complex designs, you can get faster runtime by setting a positive integer value for this variable, such as 25000. This selects the "path enumeration" mode. In this mode, to specify whether graph-based analysis paths are included in the results from exhaustive path-based analysis after the endpoint path limit is reached, set the **pba_path_recalculation_limit_compatibility** variable.

For more information, see SolvNet article 2694662, "All-Paths Exhaustive Path-Based Analysis Mode (pba_exhaustive_endpoint_path_limit = infinity)."

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

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

When an advanced on-chip variation (OCV) analysis is being performed, there are several additional variable settings that improve the runtime of the analysis.

- Enable CRPR
- Enable graph-based advanced OCV
- Enable path-based advanced OCV only mode

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
pba_derate_only_mode(3)
pba_path_recalculation_limit_compatibility(3)
timing_aocvm_enable_analysis(3)
timing_remove_clock_reconvergence_pessimism(3)
timing_report_use_worst_parallel_cell_arc(3)
```

pba_path_mode_enumerate_by_gba_slack

Specifies how paths are enumerated when you use the **-pba_mode path** option.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable applies to path-based analysis when you use the **report_timing** or **get_timing_paths** command with the **-pba_mode path** and **max_paths** options.

- **true** (the default) - Enumerates paths based on the graph-based analysis slack.

  For example, if you use the **report_timing -pba_mode path -max_paths 5 -slack_lesser_than 0** command, the tool recalculates the worst graph-based analysis path at the 5 worst violating graph-based analysis endpoints.

- **false** - Enumerates paths based on the path-based analysis slack. The tool continues to recalculate paths until the **-max_paths** option criteria is satisfied. This setting leads to a longer runtime but potentially returns a better set of relevant paths.

  For example, if you use the **report_timing -pba_mode path -max_paths 5 -slack_lesser_than 0** command, the tool recalculates the worst graph-based analysis path at the worst graph-based analysis endpoints until it finds 5 violating path-based analysis paths.

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
```

pba_path_mode_sort_by_gba_slack

Specifies how paths are sorted when you use the **-pba_mode path** option.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable applies to path-based analysis when you use the **report_timing** or **get_timing_paths** command with the **-pba_mode path** option.

When the variable is set to **false** (the default), the tool sorts and filters paths based on the recalculated slack. For example, if you use the **report_timing -slack_lesser_than 0 -pba_mode path** command, and the worst path has a graph-based slack of -1 and a recalculated slack of 1, this path does not appear in the final report.

When you set the variable to **true**, the tool sorts and filters paths based on the graph-based slack calculated during the **update_timing** command. With this setting, the **-pba_mode path** option does not change the order of the paths generated by the **report_timing** or **get_timing_paths** command.

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
```

pba_path_recalculation_limit_compatibility

Controls whether graph-based analysis paths are included in results from path-based analysis recalculation when the path recalculation limit is reached.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If this variable is set to **true** (the default), PrimeTime only returns path-based analysis paths from the **get_timing_paths** command if the **-pba_mode** option is not set to **none**. Also, when this variable is **true**, the **report_timing** command with the **-pba_mode** option set to anything other than **none** reports only recalculated paths.

If this variable is set to **false**, PrimeTime can report both recalculated and non-recalculated paths.

This variable has an effect only when the **pba_exhaustive_endpoint_path_limit** variable is set to a numeric value, which selects the "path enumeration" exhaustive path-based analysis mode and specifies the per-endpoint limit. It has no effect when the **pba_exhaustive_endpoint_path_limit** variable is set to the string "infinity", which selects the "all paths" mode, the default.

SEE ALSO

`get_timing_paths(2)`

```
report_timing(2)
pba_exhaustive_endpoint_path_limit(3)
```

pba_recalculate_full_path

Allows regular path-based analysis to recalculate full clock paths, borrowing path segments and data check reference paths.

TYPE

integer

DEFAULT

false

DESCRIPTION

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

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

In advanced on-chip variation (AOCV) mode, the depth and distance metrics are always recomputed by path recalculation regardless of the value of this variable. In advanced OCV mode, this variable only has an effect when path slew recomputation is enabled when the **pba_aocvm_only_mode** variable is set to its default of **false**. When this variable is set to **true**, path-specific delay calculation is performed along the clock and data paths. When this variable is set to **false**, path-specific delay calculation is only

performed along the data path.

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
```

pg_allow_pg_pin_reconnection

Enables reconnection of a supply net to a PG pin in the PG connectivity from netlist.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable reconnection of a supply net to a PG pin that already has a connection in the PG connectivity from netlist.

This variable allows reconnections on only PG pins. Reconnection of a supply net to a port is not allowed even with the variable set to **true**.

SEE ALSO

`connect_supply_net(2)`
`link_keep_pg_connectivity(3)`

pg_pin_info_attributes

Describes the predefined application attributes for pg_pin_info objects.

DESCRIPTION

pin_name

Type: string
Returns the pin name.

supply_connection

Type: string
Returns the supply collection.

type

Type: string
Returns primary_power or primary_ground.

voltage_for_max_delay

Type: float
Returns the voltage for maximum delay.

voltage_for_min_delay

Type: float
Returns the voltage for minimum delay.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

pin_attributes

Describes the predefined application attributes for pin objects.

DESCRIPTION

actual_fall_transition_max

Type: float
Returns the largest falling transition time for the pin.

actual_fall_transition_min

Type: float
Returns the smallest falling transition time for the pin.

actual_min_clock_pulse_width_high

Type: string
Returns the per-clock actual minimum pulse width value at the pin (high pulse).

actual_min_clock_pulse_width_low

Type: string
Returns the per-clock actual minimum pulse width value at the pin (low pulse).

actual_rise_transition_max

Type: float
Returns the largest rising transition time for the pin.

actual_rise_transition_min

Type: float
Returns the smallest rising transition time for the pin.

actual_transition_max

Type: float
Returns the maximum of the actual_rise_transition_max and actual_fall_transition_max attributes.

actual_transition_min

Type: float
Returns the minimum of the actual_rise_transition_min and actual_fall_transition_min attributes.

annotated_fall_transition_delta_max

Type: float
Returns the additional transition time added to the maximum falling transition time on the pin. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_fall_transition_delta_min

Type: float
Returns the additional transition time added to the minimum falling transition time on the pin. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_rise_transition_delta_max

Type: float
Returns the additional transition time added to the maximum rising transition time on the pin. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_rise_transition_delta_min

Type: float
Returns the additional transition time added to the minimum rising transition time on the pin. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_transition_delta_max

Type: float
Returns the maximum of the annotated_rise_transition_delta_max and annotated_fall_transition_delta_max attributes.

annotated_transition_delta_min

Type: float
Returns the minimum of the annotated_rise_transition_delta_min and annotated_fall_transition_delta_min attributes.

arrival_window

Type: string
Returns the minimum and maximum arrivals for rise and fall transitions. To get the arrival_window attribute on pins that are not endpoints, set the timing_save_pin_arrival_and_slack variable to true.

cached_c1_max_fall

Type: float
Returns the C1 CCS receiver model for a maximum fall transition.

cached_c1_max_rise

Type: float
Returns the C1 CCS receiver model for a maximum rise transition.

cached_c1_min_fall

Type: float
Returns the C1 CCS receiver model for a minimum fall transition.

cached_c1_min_rise

Type: float
Returns the C1 CCS receiver model for a minimum rise transition.

cached_c2_max_fall

Type: float
Returns the C2 CCS receiver model for a maximum fall transition.

cached_c2_max_rise

Type: float
Returns the C2 CCS receiver model for a maximum rise transition.

cached_c2_min_fall

Type: float
Returns the C2 CCS receiver model for a minimum fall transition.

cached_c2_min_rise

Type: float
Returns the C2 CCS receiver model for a minimum rise transition.

cached_ceff_max_fall

Type: float
Returns the worst effective capacitance of a net with detailed parasitics stored during the delay calculation that is connected to this output pin. This requires the rc_cache_min_max_rise_fall_ceff variable to be set to true before the timing update.

cached_ceff_max_rise

Type: float
Returns the worst effective capacitance of a net with detailed parasitics stored during the delay calculation that is connected to this output pin. This requires the rc_cache_min_max_rise_fall_ceff variable to be set to true before the timing update.

cached_ceff_min_fall

Type: float
Returns the worst effective capacitance of a net with detailed parasitics stored during the delay

calculation that is connected to this output pin. This requires the rc_cache_min_max_rise_fall_ceff variable to be set to true before the timing update.

cached_ceff_min_rise

Type: float
Returns the worst effective capacitance of a net with detailed parasitics stored during the delay calculation that is connected to this output pin. This requires the rc_cache_min_max_rise_fall_ceff variable to be set to true before the timing update.

case_value

Type: string
Returns the user-specified logic value of the pin or port propagated from a case analysis or logic constant. This attribute is computed only for leaf pins.

ceff_params_max

Type: string
Returns the parameters internally computed by PrimeTime to compute the effective capacitance of a driver pin of a cell, respectively for maximum operating conditions. The returned parameters are rd, t0, delta_t, and Ceff; they represent how a driver is modeled for computing the effective capacitance.

ceff_params_min

Type: string
Returns the parameters internally computed by PrimeTime to compute the effective capacitance of a driver pin of a cell, respectively for minimum operating conditions. The returned parameters are rd, t0, delta_t, and Ceff; they represent how a driver is modeled for computing the effective capacitance.

cell

Type: collection
Returns a collection that contains the cell that this pin belongs to.

clock_capture_arrival_dynamic

Type: string
Returns the arrival time on a clock capture path endpoint including the dynamic arrival time. The dynamic arrival time is modeled using the set_clock_latency -dynamic command. If the clock source latency is modeled including the dynamic latency (set_clock_latency -source -dynamic), the static arrival attributes exclude the dynamic value. To query this attribute, you need to enable PrimeTime SI analysis.

clock_capture_arrival_static

Type: string
Returns the static arrival time on a clock capture path endpoint. The tool calculates the static arrival time as the sum of the clock source latency (specified by the set_clock_latency -source command) and the clock network latency.

clock_latency_fall_max

Type: float
Returns the user-specified maximum fall latency (insertion delay) of a pin in the clock network. Set with the set_clock_latency command.

clock_latency_fall_min

Type: float
Returns the user-specified minimum fall latency (insertion delay) of a pin in the clock network. Set with the set_clock_latency command.

clock_latency_rise_max

Type: float
Returns the user-specified maximum rise latency (insertion delay) of a pin in the clock network. Set with set_clock_latency.

clock_latency_rise_min

Type: float
Returns the user-specified minimum rise latency (insertion delay) of a pin in the clock network. Set with set_clock_latency.

clock_launch_arrival_dynamic

Type: string
Returns the arrival time on a clock launch path endpoint including the dynamic arrival time. The dynamic arrival time is modeled using the set_clock_latency -dynamic command. If the clock source latency is modeled including the dynamic latency (set_clock_latency -source -dynamic), the static arrival attributes exclude the dynamic value. To query this attribute, you need to enable PrimeTime SI analysis.

clock_launch_arrival_static

Type: string
Returns the static arrival time on a clock launch path endpoint. The tool calculates the static arrival time as the sum of the clock source latency (specified by the set_clock_latency -source command) and the clock network latency.

clock_source_latency_early_fall_max

Type: float
Returns the maximum early falling source latency. Set with the set_clock_latency command.

clock_source_latency_early_fall_min

Type: float
Returns the minimum early falling source latency. Set with the set_clock_latency command.

clock_source_latency_early_rise_max

Type: float
Returns the maximum early rising source latency. Set with the set_clock_latency command.

clock_source_latency_early_rise_min

Type: float
Returns the minimum early rising source latency. Set with the set_clock_latency command.

clock_source_latency_late_fall_max

Type: float
Returns the maximum late falling source latency. Set with the set_clock_latency command.

clock_source_latency_late_fall_min

Type: float
Returns the minimum late falling source latency. Set with the set_clock_latency command.

clock_source_latency_late_rise_max

Type: float
Returns the maximum late rising source latency. Set with the set_clock_latency command.

clock_source_latency_late_rise_min

Type: float
Returns the minimum late rising source latency. Set with the set_clock_latency command.

clocks

Type: collection
Returns a collection of clock objects that propagate through the pin. It is undefined if no clocks are present.

constant_value

Type: string
Returns the logic value of a pin tied to logic constant zero or one in the netlist.

constraining_max_transition

Type: float
Returns the most constraining user-defined maximum transition value at a pin.

drc_constraining_max_capacitance

Type: float
Returns the user-defined maximum capacitance constraint value applicable at the the net's driver pin(s) for DRC checking.

drc_actual_max_capacitance

Type: float
Returns the actual capacitance value of a net, defined at the driver pin(s) for maximum condition DRC checking purposes. This is the value used by report_constraint.

drc_actual_max_transition

Type: float
Returns the actual transition value at the pin for maximum condition DRC checking purposes. This is the value used by report_constraint.

direction

Type: string
Returns the direction of the pin. Value can be in, out, inout, or internal. The pin_direction attribute is a synonym for direction. Directions can change as a result of linking a design, as references are resolved.

disable_timing

Type: boolean
Returns true for a disabled timing arc. This has the same effect on timing as not having the arc in the library. Set with the set_disable_timing command.

driver_model_scaling_libs_max

Type: collection
Returns a collection of library objects used for driver model scaling, where applicable, for libs maximum analysis.

driver_model_scaling_libs_min

Type: collection
Returns a collection of library objects used for driver model scaling, where applicable, for libs minimum analysis.

driver_model_type_max_fall

Type: string
Returns the driver model type, either basic (NLDM) or advanced (CCS timing), for maximum fall analysis.

driver_model_type_max_rise

Type: string
Returns the driver model type, either basic (NLDM) or advanced (CCS timing), for maximum rise analysis.

driver_model_type_min_fall

Type: string
Returns the driver model type, either basic (NLDM) or advanced (CCS timing), for minimum fall analysis.

driver_model_type_min_rise

Type: string
Returns the driver model type, either basic (NLDM) or advanced (CCS timing), for minimum rise analysis.

effective_capacitance_max

Type: float
Returns the effective capacitance for maximum operating conditions. Valid only for driver pins attached to annotated RC networks.

effective_capacitance_min

Type: float
Returns the effective capacitance for minimum operating conditions. Valid only for driver pins attached to annotated RC networks.

escaped_full_name

Type: string
Returns the name of the cell. Any literal hierarchy characters are escaped with a backslash.

fanout_load

Type: float
Returns the fanout load value of a pin. This value is used in computing max_fanout design rule cost.

full_name

Type: string
Returns the complete name of the pin to the top of the hierarchy. For example, the full name of pin Z on cell U2 within cell U1 is U1/U2/Z. The setting of the current instance has no effect on the full name of a pin. See also the lib_pin_name attribute.

glitch_rate

Type: double
Returns the rate at which the glitch transitions occur on the pin within a specific time period.

has_model_mismatch_clock

Type: boolean
Returns true if at least one model arrival or required time refers to an unmapped clock. This HyperScale attribute applies to a side-input or stub pin.

hold_uncertainty

Type: float
Returns the clock uncertainty (skew) of a clock used for hold (and other minimum delay) timing checks. Set with the set_clock_uncertainty command.

ideal_latency_max_fall

Type: float
Returns the ideal delay value annotated on a pin in an ideal network, using the set_ideal_latency command.

ideal_latency_max_rise

Type: float
Returns the ideal delay value annotated on a pin in an ideal network, using the set_ideal_latency command.

ideal_latency_min_fall

Type: float
Returns the ideal delay value annotated on a pin in an ideal network, using the set_ideal_latency command.

ideal_latency_min_rise

Type: float
Returns the ideal delay value annotated on a pin in an ideal network, using the set_ideal_latency command.

ideal_transition_max_fall

Type: float
Returns the ideal transition time annotated on a pin in an ideal network, using the set_ideal_transition command.

ideal_transition_max_rise

Type: float
Returns the ideal transition time annotated on a pin in an ideal network, using the set_ideal_transition command.

ideal_transition_min_fall

Type: float
Returns the ideal transition time annotated on a pin in an ideal network, using the set_ideal_transition command.

ideal_transition_min_rise

Type: float
Returns the ideal transition time annotated on a pin in an ideal network, using the set_ideal_transition command.

is_abstracted

Type: boolean
Returns true if the port is abstracted in the HyperScale model. This attribute applies only to ports at the block level and corresponding pins at the top level. This attribute is available only after update_timing at the block level. The attribute returns true for set_port_abstraction -ignore. Ports with is_user_abstracted == true are a subset of ports with is_abstracted == true.

is_async_pin

Type: boolean

Returns true for an asynchronous preset/clear pin.

is_clear_pin

Type: boolean
Returns true for an asynchronous clear pin.

is_clock_gating_pin

Type: boolean
Returns true for a pin of a clock-gating cell.

is_clock_network

Type: boolean
Returns true if the pin is in the combinational fanout of a clock source.

is_clock_pin

Type: boolean
Returns true on a valid instance pin object and on an active clock pin that is reached by a clock signal and where that sequential cell is not disabled by disabled timing arcs or case analysis.

is_clock_source

Type: boolean
Returns true if a design clock was declared with a source at the pin.

is_clock_source_network

Type: boolean
Returns true if the pin is part of a clock source latency network.

is_clock_used_as_clock

Type: boolean
Returns true if the clock through the pin acts as a clock.

is_clock_used_as_data

Type: boolean
Returns true if the clock through the pin acts as data.

is_data_pin

Type: boolean
Returns true if a pin is a data pin of a sequential cell. For instance pin objects, this attribute is true if the pin is a valid and active data pin that is reached by a clock signal and where that sequential cell is not disabled by disabled timing arcs or case analysis.

is_design_mismatch

Type: boolean

Returns true if the pin has a mismatch between the block and top-level design.

is_driver_scaled_max_fall

Type: boolean
Returns true if the pin uses scaling between libraries for the driver model for maximum fall analysis.

is_driver_scaled_max_rise

Type: boolean
Returns true if the pin uses scaling between libraries for the driver model for maximum rise analysis.

is_driver_scaled_min_fall

Type: boolean
Returns true if the pin uses scaling between libraries for the driver model for minimum fall analysis.

is_driver_scaled_min_rise

Type: boolean
Returns true if the pin uses scaling between libraries for the driver model for minimum rise analysis.

is_excluded

Type: boolean
Returns true if the pin is excluded from timing analysis. The pin is inside the HyperScale block and visible at HyperScale top, retained for circuit completion.

is_fall_edge_triggered_clock_pin

Type: boolean
Returns true if the pin is used as a falling-edge-triggered flip-flop clock pin.

is_fall_edge_triggered_data_pin

Type: boolean
Returns true if the pin is used as a falling-edge-triggered flip-flop data pin.

is_hierarchical

Type: boolean
Returns true for a pin that belongs to an instantiation of another design, or false for a pin that belongs to an instantiation of a library cell (a leaf pin).

is_ideal

Type: boolean
Returns true if the pin has been marked ideal using the set_ideal_network command.

is_interface_logic_pin

Type: boolean
Returns true if the pin is in an interface logic model (ILM) of the design. This attribute is read-only; you

cannot change the setting.

is_latch_loop_breaker

Type: boolean
Returns true if the pin is the D pin of a loop-breaker latch.

is_mux_select_pin

Type: boolean
Returns true if the pin is the select pin of a multiplexer device.

is_negative_level_sensitive_clock_pin

Type: boolean
Returns true if the pin is used as a negative level-sensitive latch clock pin.

is_negative_level_sensitive_data_pin

Type: boolean
Returns true if the pin is used as a negative level-sensitive latch data pin.

is_port

Type: boolean
Returns true for a pin or a port. Pins or ports are accessible only from a timing_point object.

is_positive_level_sensitive_clock_pin

Type: boolean
Returns true if the pin is used as a positive level-sensitive latch clock pin.

is_positive_level_sensitive_data_pin

Type: boolean
Returns true if the pin is used as a positive level-sensitive latch data pin.

is_preset_pin

Type: boolean
Returns true if the pin is an asynchronous preset pin.

is_receiver_scaled_max_fall

Type: boolean
Returns true if the pin uses scaling between libraries for the receiver model for maximum fall analysis.

is_receiver_scaled_max_rise

Type: boolean
Returns true if the pin uses scaling between libraries for the receiver model for maximum rise analysis.

is_receiver_scaled_min_fall

Type: boolean
Returns true if the pin uses scaling between libraries for the receiver model for minimum fall analysis.

is_receiver_scaled_min_rise

Type: boolean
Returns true if the pin uses scaling between libraries for the receiver model for minimum rise analysis.

is_rise_edge_triggered_clock_pin

Type: boolean
Returns true if the pin is used as a rising-edge-triggered flip-flop clock pin.

is_rise_edge_triggered_data_pin

Type: boolean
Returns true if the pin is used as a rising-edge-triggered flip-flop data pin.

is_side_input

Type: boolean
Returns true if the pin is inside the HyperScale block and visible at HyperScale top, where the entire fanin (starting from internal registers) is removed, and valid slews, arrivals, or case values are annotated.

is_stub

Type: boolean
Returns true if the pin is inside the HyperScale block and visible at HyperScale top, where the entire fanout (ending at internal registers) is removed, and valid required times are annotated for accurate slack computation.

is_three_state

Type: boolean
Returns true if the pin is a three-state driver.

is_three_state_enable_pin

Type: boolean
Returns true if the pin is an enable pin of a three-state device.

is_three_state_output_pin

Type: boolean
Returns true if the pin could output a three-state signal.

is_user_abstracted

Type: boolean
Returns true if the pin is abstracted because of user-specified settings (set_port_abstraction -ignore) in the HyperScale model. This attribute applies only to ports at the block level and corresponding pins at the top level. This attribute is available only after update_timing at the block level. Ports with

is_user_abstracted == true also have is_abstracted == true.

launch_clocks

Type: collection
Returns a collection of the clocks that launch signals reaching the pin.

lib_pin

Type: collection
Returns a collection of library pins for this pin; this attribute is defined only on pins of leaf cells.

lib_pin_name

Type: string
Returns the leaf pin name. For example, the lib_pin_name of pin U2/U1/Z is Z. This attribute is read-only.

max_capacitance

Type: float
Returns the maximum capacitance design rule limit for a pin.

max_fall_arrival

Type: float
Returns the arrival time for the longest path with a falling transition on a pin. In best-case, worst-case mode, this value is for the worst-case operating condition.

max_fall_delay

Type: float
Returns the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_max_delay command.

max_fall_local_slack

Type: float
For a pin of a transparent latch, the local slack is the timing slack considering the data arrival at the local pin and setup constraints at the local pin, without considering constraints downstream from the pin. If there are no constraints at the pin, the attribute is undefined. The normal slack (max_fall_slack or max_slack) considers the constraints downstream from the pin.

max_fall_slack

Type: float
Returns the worst slack at a pin for falling maximum path delays. This attribute is valid only after timing has been updated. If the **timing_save_pin_arrival_and_slack** variable is **false** (the default), the attribute is valid only for path endpoints (register data pins and primary outputs). Switching the variable to **true** allows the attribute to be queried at all pins; in this case, the worst slack among all paths traversing a pin is returned.

max_fanout

Type: float
Returns the maximum fanout design rule limit for a pin.

max_rise_arrival

Type: float
Returns the arrival time for the longest path with a rising transition on a pin. In best-case, worst-case mode, this value is for the worst-case operating condition.

max_rise_delay

Type: float
Returns the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_max_delay command.

max_rise_local_slack

Type: float
For a pin of a transparent latch, the local slack is the timing slack considering the data arrival at the local pin and setup constraints at the local pin, without considering constraints downstream from the pin. If there are no constraints at the pin, the attribute is undefined. The normal slack (max_rise_slack or max_slack) considers the constraints downstream from the pin.

max_rise_slack

Type: float
Returns the worst slack at a pin for rising maximum path delays. This attribute is valid only after timing has been updated. If the **timing_save_pin_arrival_and_slack** variable is **false** (the default), the attribute is valid only for path endpoints (register data pins and primary outputs). Switching the variable to **true** allows the attribute to be queried at all pins; in this case, the worst slack among all paths traversing a pin is returned.

max_slack

Type: float
Returns the minimum of the max_rise_slack and max_fall_slack attributes.

max_time_borrow

Type: float
Returns a floating-point number that establishes an upper limit for time borrowing; that is, it prevents the use of the entire pulse width for level-sensitive latches. Units are those used in the logic library. Set with the set_max_time_borrow command.

max_transition

Type: float
Returns the maximum transition time design rule limit for a pin.

min_capacitance

Type: float
Returns the minimum capacitance design rule limit for a pin.

min_fall_arrival

Type: float
Returns the arrival time for the shortest path with a falling transition on a pin. In best-case worst-case mode, this value is for the best-case mode.

min_fall_delay

Type: float
Returns the minimum falling delay on clocks, pins, cells, or on paths between such objects. Set with the set_min_delay command.

min_fall_slack

Type: float
Returns the worst slack at a pin for falling minimum path delays. This attribute is valid only after timing has been updated. If the **timing_save_pin_arrival_and_slack** variable is **false** (the default), the attribute is valid only for path endpoints (register data pins and primary outputs). Switching the variable to **true** allows the attribute to be queried at all pins; in this case, the worst slack among all paths traversing a pin is returned.

min_fanout

Type: float
Returns the minimum fanout design rule limit for a pin.

min_rise_arrival

Type: float
Returns the worst hold slack of all paths passing through a pin with a rising transition at a pin. In best-case, worst-case operating conditions, this value is for the best-case condition. If all such paths are unconstrained, the value is infinity.

min_rise_delay

Type: float
Returns the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_min_delay command.

min_rise_slack

Type: float
Returns the worst slack at a pin for rising minimum path delays. This attribute is valid only after timing has been updated. If the **timing_save_pin_arrival_and_slack** variable is **false** (the default), the attribute is valid only for path endpoints (register data pins and primary outputs). Switching the variable to **true** allows the attribute to be queried at all pins; in this case, the worst slack among all paths traversing a pin is returned.

min_slack

Type: float
Returns the minimum of the min_rise_slack and min_fall_slack attributes.

min_transition

Type: float
Returns the minimum transition time design rule limit for a pin.

net

Type: collection
Returns a collection that contains the net connected to this pin; this attribute is defined only if the pin is connected to a net.

object_class

Type: string
Returns the class of the object, which is "pin". You cannot set this attribute.

pg_level

Type: string
Returns PG connectivity status of the pin. A return value '1' means the pin has a connection to a power PG net or logic constant 1, whereas a return value '0' means the pin has a connection to a ground PG net or logic constant 0.

pin_capacitance_max

Type: float
Returns the capacitance of a pin for maximum conditions.

pin_capacitance_max_fall

Type: float
Returns the maximum fall capacitance of the pin. This attribute is read-only; you cannot change the setting.

pin_capacitance_max_rise

Type: float
Returns the maximum rise capacitance of the pin. This attribute is read-only; you cannot change the setting.

pin_capacitance_min

Type: float
Returns the capacitance of a pin for minimum conditions.

pin_capacitance_min_fall

Type: float
Returns the minimum fall capacitance of the pin. This attribute is read-only; you cannot change the setting.

pin_capacitance_min_rise

Type: float
Returns the minimum rise capacitance of the pin. This attribute is read-only; you cannot change the setting.

pin_direction

Type: string
Returns the direction of the pin. Value can be in, out, inout, or internal. This attribute exists for backward compatibility with dc_shell. See the "direction" attribute.

power_base_clock

Type: string
Returns the name of the base clock associated with this pin. If the pin belongs to multiple clock domains, the power_base_clock attribute is set to the fastest of the clocks.

power_rail_voltage_bidir_input_max

Type: float
Returns the input voltage of a bidirectional pin.

power_rail_voltage_bidir_input_min

Type: float
Returns the input voltage of a bidirectional pin.

power_rail_voltage_max

Type: float
Returns the maximum power rail voltage set on the pin. In the case of a bidirectional pin, the output voltage is returned by default. To return the input voltage of a bidirectional pin, query the power_rail_voltage_bidir_input_max attribute.

power_rail_voltage_min

Type: float
Returns the minimum power rail voltage set on the pin. In the case of a bidirectional pin, the output voltage is returned by default. To return the input voltage of a bidirectional pin, query the power_rail_voltage_bidir_input_min attribute.

propagated_clock

Type: boolean
Returns true if clock edge times are delayed by propagating the values through the clock network. If this attribute is not present, ideal clocking is assumed. Set with set_propagated_clock.

rc_input_threshold_pct_fall_max

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_input_threshold_pct_fall_min

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_input_threshold_pct_rise_max

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_input_threshold_pct_rise_min

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_output_threshold_pct_fall_max

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_output_threshold_pct_fall_min

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_output_threshold_pct_rise_max

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_output_threshold_pct_rise_min

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_derate_from_library_max

Type: float
Returns the slew derating factor that the tool uses to calculate delays and transition times. This

attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_derate_from_library_min

Type: float
Returns the slew derating factor that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_lower_threshold_pct_fall_max

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_lower_threshold_pct_fall_min

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_lower_threshold_pct_rise_max

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_lower_threshold_pct_rise_min

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_upper_threshold_pct_fall_max

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_upper_threshold_pct_fall_min

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_upper_threshold_pct_rise_max

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

rc_slew_upper_threshold_pct_rise_min

Type: float
Returns the characterization trip point (waveform measurement threshold) that the tool uses to calculate delays and transition times. This attribute on an instance pin returns the library threshold value obtained from the maximum library in a min-max analysis.

receiver_model_scaling_libs_max

Type: collection
Returns a collection of library objects used for receiver model scaling, where applicable, for libs maximum analysis.

receiver_model_scaling_libs_min

Type: collection
Returns a collection of library objects used for receiver model scaling, where applicable, for libs minimum analysis.

receiver_model_type_max_fall

Type: string
Returns the receiver model type, basic (NLDM) or advanced (CCS timing), for maximum fall analysis.

receiver_model_type_max_rise

Type: string
Returns the receiver model type, basic (NLDM) or advanced (CCS timing), for maximum rise analysis.

receiver_model_type_min_fall

Type: string
Returns the receiver model type, basic (NLDM) or advanced (CCS timing), for minimum fall analysis.

receiver_model_type_min_rise

Type: string
Returns the receiver model type, basic (NLDM) or advanced (CCS timing), for minimum rise analysis.

relative_toggle_rate

Type: double
Returns the number of toggles per period of the power_base_clock; if the design does not have a defined clock, this attribute is undefined.

setup_uncertainty

Type: float

Returns the clock uncertainty (skew) of a clock used for setup (and other maximum delay) timing checks. Set with set_clock_uncertainty.

si_noise_active_aggressors_above_high

Type: collection
Returns a collection of a subset of effective aggressors that contributed to the worst case alignment to have largest effect on noise bump height.

si_noise_active_aggressors_above_low

Type: collection
Returns a collection of a subset of effective aggressors that contributed to the worst case alignment to have largest effect on noise bump height.

si_noise_active_aggressors_below_high

Type: collection
Returns a collection of a subset of effective aggressors that contributed to the worst case alignment to have largest effect on noise bump height.

si_noise_active_aggressors_below_low

Type: collection
Returns a collection of a subset of effective aggressors that contributed to the worst case alignment to have largest effect on noise bump height.

si_noise_bumps_above_high

Type: string
Returns a string that lists the aggressor nets for the input pin and their corresponding coupled bump heights and widths, considering noise bumps in the above-high region. The format of the string is:

```
{{aggr1_name height width}
 {aggr2_name height width}
  ... }
```

Height is in volts and width is in library time units.

si_noise_bumps_above_low

Type: string
Returns a string that lists the aggressor nets for the input pin and their corresponding coupled bump heights and widths, considering noise bumps in the above-low region. The format of the string is:

```
{{aggr1_name height width}
 {aggr2_name height width}
  ... }
```

Height is in volts and width is in library time units.

si_noise_bumps_below_high

Type: string
Returns a string that lists the aggressor nets for the input pin and their corresponding coupled bump

heights and widths, considering noise bumps in the below-high region. The format of the string is:

```
{{aggr1_name height width}
 {aggr2_name height width}
  ... }
```

Height is in volts and width is in library time units.

si_noise_bumps_below_low

Type: string
Returns a string that lists the aggressor nets for the input pin and their corresponding coupled bump heights and widths, considering noise bumps in the below-low region. The format of the string is:

```
{{aggr1_name height width}
 {aggr2_name height width}
  ... }
```

Height is in volts and width is in library time units.

si_noise_height_factor_above_high

Type: float
Returns the noise height derating factor for the pin in the above-high region, as set by the **set_noise_derate** command.

si_noise_height_factor_above_low

Type: float
Returns the noise height derating factor for the pin in the above-low region.

si_noise_height_factor_below_high

Type: float
Returns the noise height derating factor for the pin in the below-high region.

si_noise_height_factor_below_low

Type: float
Returns the noise height derating factor for the pin in the below-low region.

si_noise_height_offset_above_high

Type: float
Returns the noise height offset for the pin in the above-high region, as set by the **set_noise_derate** command.

si_noise_height_offset_above_low

Type: float
Returns the noise height offset for the pin in the above-low region.

si_noise_height_offset_below_high

Type: float

Returns the noise height offset for the pin in the below-high region.

si_noise_height_offset_below_low

Type: float
Returns the noise height offset for the pin in the below-low region.

si_noise_lib_pin_name

Type: string
Returns the name of the library pin of equivalent library cell specified for noise analysis.

si_noise_prop_bumps_above_high

Type: string
Returns a string that shows the bump height and width at the input pin caused by noise propagation in the above-high region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_prop_bumps_above_low

Type: string
Returns a string that shows the bump height and width at the input pin caused by noise propagation in the above-low region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_prop_bumps_below_high

Type: string
Returns a string that shows the bump height and width at the input pin caused by noise propagation in the below-high region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_prop_bumps_below_low

Type: string
Returns a string that shows the bump height and width at the input pin caused by noise propagation in the below-low region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_slack_above_high

Type: float
Returns the amount of noise slack for the pin in the above-high region.

si_noise_slack_above_low

Type: float
Returns the amount of noise slack for the pin in the above-low region.

si_noise_slack_below_high

Type: float

Returns the amount of noise slack for the pin in the below-high region.

si_noise_slack_below_low

Type: float
Returns the amount of noise slack for the pin in the below-low region.

si_noise_total_bump_above_high

Type: string
Returns a string that shows the total bump height and width at the input pin caused by crosstalk and noise propagation in the above-high region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_total_bump_above_low

Type: string
Returns a string that shows the total bump height and width at the input pin caused by crosstalk and noise propagation in the above-low region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_total_bump_below_high

Type: string
Returns a string that shows the total bump height and width at the input pin caused by crosstalk and noise propagation in the below-high region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_total_bump_below_low

Type: string
Returns a string that shows the total bump height and width at the input pin caused by crosstalk and noise propagation in the below-low region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_width_factor_above_high

Type: float
Returns the noise width derating factor for the pin in the above-high region.

si_noise_width_factor_above_low

Type: float
Returns the noise width derating factor for the pin in the above-low region.

si_noise_width_factor_below_high

Type: float
Returns the noise width derating factor for the pin in the below-high region.

si_noise_width_factor_below_low

Type: float

Returns the noise width derating factor for the pin in the below-low region.

si_noise_worst_prop_arc_above_high

Type: collection
Returns the cell arc that corresponds to the worst noise propagation to an output pin of a cell.

si_noise_worst_prop_arc_above_low

Type: collection
Returns the cell arc that corresponds to the worst noise propagation to an output pin of a cell.

si_noise_worst_prop_arc_below_high

Type: collection
Returns the cell arc that corresponds to the worst noise propagation to an output pin of a cell.

si_noise_worst_prop_arc_below_low

Type: collection
Returns the cell arc that corresponds to the worst noise propagation to an output pin of a cell.

static_probability

Type: float
Returns the static probability that the pin has the logic value 1.

temperature_max

Type: float
Returns the maximum temperature for the cell. This value is set by the operating condition specification or the set_temperature command.

temperature_min

Type: float
Returns the minimum temperature for the cell. This value is set by the operating condition specification or the set_temperature command.

toggle_rate

Type: double
Returns the rate at which transitions occur on the pin within a time period.

user_case_value

Type: string
Returns the user-specified logic value of a pin or port.

worst_max_time_borrow_fall

Type: float
Returns the maximum time that could be potentially borrowed across all incoming paths to the latch

and across all capturing clocks at the latch. The value is the most restrictive of the following values:

- Closing edge
- Value specified by the **set_max_time_borrow** command

The attribute returns

- Zero if the endpoint is not a transparent latch D pin
- Path corner of the distribution of the actual variation quantity if the design uses parametric on-chip variation (POCV)

worst_max_time_borrow_rise

Type: float
Returns the maximum time that could be potentially borrowed across all incoming paths to the latch and across all capturing clocks at the latch. The value is the most restrictive of the following values:

- Closing edge
- Value specified by the **set_max_time_borrow** command

The attribute returns

- Zero if the endpoint is not a transparent latch D pin
- Path corner of the distribution of the actual variation quantity if the design uses parametric on-chip variation (POCV)

x_coordinate

Type: float
Returns the x-coordinate of the pin.

y_coordinate

Type: float
Returns the y-coordinate of the pin.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

port_attributes

Describes the predefined application attributes for port objects.

DESCRIPTION

actual_fall_transition_max

Type: float
Returns the largest falling transition time for the port. You cannot set this attribute.

actual_fall_transition_min

Type: float
Returns the smallest falling transition time for the port. You cannot set this attribute.

actual_min_clock_pulse_width_high

Type: string
Returns a string containing a per-clock actual minimum pulse width value at the port (high pulse).

actual_min_clock_pulse_width_low

Type: string
Returns a string containing a per-clock actual minimum pulse width value at the port (low pulse).

actual_rise_transition_max

Type: float
Returns the largest rising transition time for the port. You cannot set this attribute.

actual_rise_transition_min

Type: float
Returns the smallest rising transition time for the port. You cannot set this attribute.

actual_transition_max

Type: float
Returns the maximum of the actual_rise_transition_max and actual_fall_transition_max attributes.

actual_transition_min

Type: float
Returns the minimum of the actual_rise_transition_min and actual_fall_transition_min attributes.

annotated_fall_transition_delta_max

Type: float
Returns the additional transition time added to the maximum falling transition time on the port. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_fall_transition_delta_min

Type: float
Returns the additional transition time added to the minimum falling transition time on the port. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_rise_transition_delta_max

Type: float
Returns the additional transition time added to the maximum rising transition time on the port. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_rise_transition_delta_min

Type: float
Returns the additional transition time added to the minimum rising transition time on the port. The additional transition time is set by either the set_annotated_transition -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_transition_delta_max

Type: float
Returns the maximum of the annotated_rise_transition_delta_max and annotated_fall_transition_delta_max attributes.

annotated_transition_delta_min

Type: float
Returns the minimum of the annotated_rise_transition_delta_min and annotated_fall_transition_delta_min attributes.

arrival_window

Type: string
Returns the minimum and maximum arrivals for rising and falling transitions.

cached_c1_max_fall

Type: float
Returns the value of C1 (first capacitance of CCS receiver model) stored during the delay calculation.

cached_c1_max_rise

Type: float
Returns the value of C1 (first capacitance of CCS receiver model) stored during the delay calculation.

cached_c1_min_fall

Type: float
Returns the value of C1 (first capacitance of CCS receiver model) stored during the delay calculation.

cached_c1_min_rise

Type: float
Returns the value of C1 (first capacitance of CCS receiver model) stored during the delay calculation.

cached_c2_max_fall

Type: float
Returns the value of C2 (second capacitance of CCS receiver model) stored during the delay calculation.

cached_c2_max_rise

Type: float
Returns the value of C2 (second capacitance of CCS receiver model) stored during the delay calculation.

cached_c2_min_fall

Type: float
Returns the value of C2 (second capacitance of CCS receiver model) stored during the delay calculation.

cached_c2_min_rise

Type: float
Returns the value of C2 (second capacitance of CCS receiver model) stored during the delay calculation.

cached_ceff_max_fall

Type: float
Worst effective capacitance of a net with detailed parasitics stored during the delay calculation that is connected to this input port. This requires the rc_cache_min_max_rise_fall_ceff variable to be set to true before the timing update.

cached_ceff_max_rise

Type: float
Worst effective capacitance of a net with detailed parasitics stored during the delay calculation that is connected to this input port. This requires the rc_cache_min_max_rise_fall_ceff variable to be set to true before the timing update.

cached_ceff_min_fall

Type: float
Worst effective capacitance of a net with detailed parasitics stored during the delay calculation that is connected to this input port. This requires the rc_cache_min_max_rise_fall_ceff variable to be set to true before the timing update.

cached_ceff_min_rise

Type: float
Worst effective capacitance of a net with detailed parasitics stored during the delay calculation that is connected to this input port. This requires the rc_cache_min_max_rise_fall_ceff variable to be set to true before the timing update.

case_value

Type: string
Returns the user-specified logic value of a pin or port propagated from a case analysis or logic constant.

ceff_params_max

Type: string
Returns the parameters used to find the maximum effective capacitance for each timing arc feeding a driver pin attached to an annotated RC network. The parameters specify a linear driver model (rd = drive resistance, t0 = start of voltage ramp, delta_t = duration of voltage ramp, ceff = effective capacitance).

ceff_params_min

Type: string
Returns the parameters used to find the minimum effective capacitance for each timing arc feeding a driver pin attached to an annotated RC network. The parameters specify a linear driver model (rd = drive resistance, t0 = start of voltage ramp, delta_t = duration of voltage ramp, ceff = effective capacitance).

clock_capture_arrival_dynamic

Type: string
Returns the arrival time on a clock capture path endpoint including the dynamic arrival time. The dynamic arrival time is modeled using the set_clock_latency -dynamic command. If the clock source latency is modeled including the dynamic latency (set_clock_latency -source -dynamic), the static arrival attributes exclude the dynamic value. To query this attribute, you need to enable PrimeTime SI analysis.

clock_capture_arrival_static

Type: string
Returns the static arrival time on a clock capture path endpoint. The tool calculates the static arrival time as the sum of the clock source latency (specified by the set_clock_latency -source command) and the clock network latency.

clock_latency_fall_max

Type: float
Returns the user-specified maximum fall latency (insertion delay) for clock networks through the port. Set with the set_clock_latency command.

clock_latency_fall_min

Type: float
Returns the user-specified minimum fall latency (insertion delay) for clock networks through the port. Set with the set_clock_latency command.

clock_latency_rise_max

Type: float
Returns the user-specified maximum rise latency (insertion delay) for clock networks through the port. Set with the set_clock_latency command.

clock_latency_rise_min

Type: float
Returns the user-specified minimum rise latency (insertion delay) for clock networks through the port. Set with the set_clock_latency command.

clock_launch_arrival_dynamic

Type: string
Returns the arrival time on a clock launch path endpoint including the dynamic arrival time. The dynamic arrival time is modeled using the set_clock_latency -dynamic command. If the clock source latency is modeled including the dynamic latency (set_clock_latency -source -dynamic), the static arrival attributes exclude the dynamic value. To query this attribute, you need to enable PrimeTime SI analysis.

clock_launch_arrival_static

Type: string
Returns the static arrival time on a clock launch path endpoint. The tool calculates the static arrival time as the sum of the clock source latency (specified by the set_clock_latency -source command) and the clock network latency.

clock_source_latency_early_fall_max

Type: float
Returns the maximum early falling source latency. Set with the set_clock_latency command.

clock_source_latency_early_fall_min

Type: float
Returns the minimum early falling source latency. Set with the set_clock_latency command.

clock_source_latency_early_rise_max

Type: float

Returns the maximum early rising source latency. Set with the set_clock_latency command.

clock_source_latency_early_rise_min

Type: float
Returns the minimum early rising source latency. Set with the set_clock_latency command.

clock_source_latency_late_fall_max

Type: float
Returns the maximum late falling source latency. Set with the set_clock_latency command.

clock_source_latency_late_fall_min

Type: float
Returns the minimum late falling source latency. Set with the set_clock_latency command.

clock_source_latency_late_rise_max

Type: float
Returns the maximum late rising source latency. Set with the set_clock_latency command.

clock_source_latency_late_rise_min

Type: float
Returns the minimum late rising source latency. Set with the set_clock_latency command.

clocks

Type: collection
Returns a collection of clock objects which propagate through this port. It is undefined if no clocks are present.

connection_class

Type: string
Returns the connection class string for the port.

constant_value

Type: string
Returns the logic value of the port tied to logic constant zero or one in the netlist.

constraining_max_transition

Type: float
Returns the most constraining user-defined maximum transition value at the port.

drc_constraining_max_capacitance

Type: float
Returns the user-defined maximum capacitance constraint value applicable at the the net's driver pin(s) of the port for DRC checking.

drc_actual_max_capacitance

Type: float
Returns the actual capacitance value of a net, defined at the driver pin(s) of the port for maximum condition DRC checking purposes. This is the value used by report_constraint.

drc_actual_max_transition

Type: float
Returns the actual transition value at the port for maximum condition DRC checking purposes. This is the value used by report_constraint.

direction

Type: string
Returns the direction of the port. Value can be in, out, inout, or internal. The port_direction attribute is a synonym for direction. You cannot set this attribute.

disable_timing

Type: boolean
Returns true if the timing for the port has been marked as disabled with the set_disable_timing command.

drive_resistance_fall_max

Type: float
Returns the linear drive resistance for falling delays and maximum conditions, associated with an input or inout port. Set with the set_drive command.

drive_resistance_fall_min

Type: float
Returns the linear drive resistance for falling delays and minimum conditions, associated with an input or inout port. Set with the set_drive command.

drive_resistance_rise_max

Type: float
Returns the linear drive resistance for rising delays and maximum conditions, associated with an input or inout port. Set with the set_drive command.

drive_resistance_rise_min

Type: float
Returns the linear drive resistance for rising delays and minimum conditions, associated with an input or inout port. Set with the set_drive command.

driver_model_scaling_libs_max

Type: collection
Returns a collection of library objects used for driver model scaling, where applicable, for maximum analysis.

driver_model_scaling_libs_min

Type: collection
Returns a collection of library objects used for driver model scaling, where applicable, for minimum analysis.

driver_model_type_max_fall

Type: string
Returns the driver model type, basic (NLDM) or advanced (CCS timing), for maximum fall analysis.

driver_model_type_max_rise

Type: string
Returns the driver model type, basic (NLDM) or advanced (CCS timing), for maximum rise analysis.

driver_model_type_min_fall

Type: string
Returns the driver model type, basic (NLDM) or advanced (CCS timing), for minimum fall analysis.

driver_model_type_min_rise

Type: string
Returns the driver model type, basic (NLDM) or advanced (CCS timing), for minimum rise analysis.

driving_cell_fall_max

Type: string
Returns a library cell from which to copy maximum fall drive capability to be used in fall transition calculation for the port. Set with the set_driving_cell command.

driving_cell_fall_min

Type: string
Returns a library cell from which to copy the minimum fall drive capability to be used in fall transition calculation for the port. Set with the set_driving_cell command.

driving_cell_from_pin_fall_max

Type: string
Returns the driving_cell_fall_max input pin to be used to find timing arc maximum fall drive capability. Set with the set_driving_cell command.

driving_cell_from_pin_fall_min

Type: string
Returns the driving_cell_fall_min input pin to be used to find timing arc minimum fall drive capability. Set with the set_driving_cell command.

driving_cell_from_pin_rise_max

Type: string

Returns the driving_cell_rise_max input pin to be used to find timing arc rise drive capability. Set with the set_driving_cell command.

driving_cell_from_pin_rise_min

Type: string
Returns the driving_cell_rise_min input pin to be used to find timing arc rise drive capability. Set with the set_driving_cell command.

driving_cell_library_fall_max

Type: string
Returns the library in which to find the driving_cell_fall_max. Set with the set_driving_cell command.

driving_cell_library_fall_min

Type: string
Returns the library in which to find the driving_cell_fall_min. Set with the set_driving_cell command.

driving_cell_library_rise_max

Type: string
Returns the library in which to find the driving_cell_rise_max. Set with the set_driving_cell command.

driving_cell_library_rise_min

Type: string
Returns the library in which to find the driving_cell_rise_min. Set with the set_driving_cell command.

driving_cell_max_fall_itrans_fall

Type: float
Returns the value of the maximum input transition for the driving cell that was associated with the port by the set_driving_cell command. This attribute represents the falling transition at the from_pin of the driving cell that is used to compute the falling transition value at a pin that drives the port.

driving_cell_max_fall_itrans_rise

Type: float
Returns the value of the maximum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the falling transition value at the from_pin of the driving cell that is used to compute the rising transition value at the pin that drives the port.

driving_cell_max_rise_itrans_fall

Type: float
Returns the value of the maximum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the rising transition value at the from_pin of the driving cell that is used to compute falling transition value at the pin that drives the port.

driving_cell_max_rise_itrans_rise

Type: float

Returns the value of the maximum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the rising transition value at the from_pin of the driving cell that is used to compute the rising transition value at the pin that drives the port.

driving_cell_min_fall_itrans_fall

Type: float
Returns the value of the minimum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the minimum falling transition value at the from_pin of the driving cell that is used to compute the falling transition value at the pin that drives the port.

driving_cell_min_fall_itrans_rise

Type: float
Returns the value of the minimum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the minimum falling transition value at the from_pin of the driving cell that is used to compute the rising transition value at the pin that drives the port.

driving_cell_min_rise_itrans_fall

Type: float
Returns the value of the minimum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the minimum rise transition value at the from_pin of the driving cell that is used to compute the falling transition value at the pin that drives the port.

driving_cell_min_rise_itrans_rise

Type: float
Returns the value of the minimum input transition for the driving cell that was associated with a port by the set_driving_cell command. This attribute represents the minimum rise transition value at the from_pin of the driving cell that is used to compute the rising transition value at the pin that drives the port.

driving_cell_no_design_rule

Type: boolean
Returns true if driving cell information has been set on a port with set_driving_cell -no_design_rule. If true, the driving cell's design rule limits (max_capacitance and so forth) are not used for the port.

driving_cell_pin_fall_max

Type: string
Returns the driving_cell_fall_max output pin to be used to find timing arc fall drive capability. Set with the set_driving_cell command.

driving_cell_pin_fall_min

Type: string
Returns the driving_cell_fall_min output pin to be used to find timing arc fall drive capability. Set with the set_driving_cell command.

driving_cell_pin_rise_max

Type: string
Returns the driving_cell_rise_max output pin to be used to find timing arc rise drive capability. Set with the set_driving_cell command.

driving_cell_pin_rise_min

Type: string
Returns the driving_cell_rise_min output pin to be used to find timing arc rise drive capability. Set with the set_driving_cell command.

driving_cell_rise_max

Type: string
Returns a library cell from which to copy maximum rise drive capability to be used in rise transition calculation for the port. Set with the set_driving_cell command.

driving_cell_rise_min

Type: string
Returns a library cell from which to copy the minimum rise drive capability to be used in rise transition calculation for the port. Set with the set_driving_cell command.

effective_capacitance_max

Type: float
Returns the effective capacitance for maximum operating conditions. Valid only for driver pins attached to annotated RC networks.

effective_capacitance_min

Type: float
Returns the effective capacitance for minimum operating conditions. Valid only for driver pins attached to annotated RC networks.

escaped_full_name

Type: string
Returns the name of the cell. Any literal hierarchy characters are escaped with a backslash.

fanout_load

Type: float
Returns the fanout load on output ports. Set with the set_fanout_load command.

full_name

Type: string
Returns the name of a port. You cannot set this attribute.

hold_uncertainty

Type: float
Returns the clock uncertainty (skew) of a clock used for hold (and other minimum delay) timing checks. Set with the set_clock_uncertainty command.

ideal_latency_max_fall

Type: float
Returns the ideal delay value annotated on a port in an ideal network. To set this value, use the set_ideal_latency command.

ideal_latency_max_rise

Type: float
Returns the ideal delay value annotated on a port in an ideal network. To set this value, use the set_ideal_latency command.

ideal_latency_min_fall

Type: float
Returns the ideal delay value annotated on a port in an ideal network. To set this value, use the set_ideal_latency command.

ideal_latency_min_rise

Type: float
Returns the ideal delay value annotated on a port in an ideal network. To set this value, use the set_ideal_latency command.

ideal_transition_max_fall

Type: float
Returns the ideal transition time annotated on a port in an ideal network. To set this value, use the set_ideal_transition command.

ideal_transition_max_rise

Type: float
Returns the ideal transition time annotated on a port in an ideal network. To set this value, use the set_ideal_transition command.

ideal_transition_min_fall

Type: float
Returns the ideal transition time annotated on a port in an ideal network. To set this value, use the set_ideal_transition command.

ideal_transition_min_rise

Type: float
Returns the ideal transition time annotated on a port in an ideal network. To set this value, use the set_ideal_transition command.

input_transition_fall_max

Type: float
Returns the fixed transition time for falling delays, maximum conditions associated with an input or inout port. Set with the set_input_transition command.

input_transition_fall_min

Type: float
Returns the fixed transition time for falling delays and minimum conditions associated with an input or inout port. Set with the set_input_transition command.

input_transition_rise_max

Type: float
Returns the fixed transition time for rising delays and maximum conditions associated with an input or inout port. Set with the set_input_transition command.

input_transition_rise_min

Type: float
Returns the fixed transition time for rising delays and minimum conditions associated with an input or inout port. Set with the set_input_transition command.

is_abstracted

Type: boolean
Returns true if the port is abstracted in the HyperScale model. This attribute applies only to ports at the block level and corresponding pins at the top level. This attribute is available only after update_timing at the block level. The attribute returns true for set_port_abstraction -ignore. Ports with is_user_abstracted == true are a subset of ports with is_abstracted == true.

is_clock_dont_override

Type: boolean
Returns true if context override is not applied because of context referring to unmapped clocks. This attribute applies only to ports at the block level.

is_clock_network

Type: boolean
Returns true if the port is a clock source or is in the combinational fanout of a clock source.

is_clock_source

Type: boolean
Returns true if a design clock was declared with a source at the port.

is_clock_source_network

Type: boolean
Returns true if the port is part of a clock source latency network.

is_clock_used_as_clock

Type: boolean
Returns true if the clock through the port acts as a clock.

is_clock_used_as_data

Type: boolean
Returns true if the clock through the port acts as data.

is_driver_scaled_max_fall

Type: boolean
Returns true if the port uses scaling between libraries for the driver model for maximum fall analysis.

is_driver_scaled_max_rise

Type: boolean
Returns true if the port uses scaling between libraries for the driver model for maximum rise analysis.

is_driver_scaled_min_fall

Type: boolean
Returns true if the port uses scaling between libraries for the driver model for minimum fall analysis.

is_driver_scaled_min_rise

Type: boolean
Returns true if the port uses scaling between libraries for the driver model for minimum rise analysis.

is_ideal

Type: boolean
Returns true if the port has been marked ideal using the set_ideal_network command.

is_netlist_dont_override

Type: boolean
Returns true if context override is not applied because of anchor leaf pin change on a port net.

is_receiver_scaled_max_fall

Type: boolean
Returns true if the port uses scaling between libraries for the receiver model for maximum fall analysis.

is_receiver_scaled_max_rise

Type: boolean
Returns true if the port uses scaling between libraries for the receiver model for maximum rise analysis.

is_receiver_scaled_min_fall

Type: boolean
Returns true if the port uses scaling between libraries for the receiver model for minimum fall analysis.

is_receiver_scaled_min_rise

Type: boolean
Returns true if the port uses scaling between libraries for the receiver model for minimum rise analysis.

is_user_abstracted

Type: boolean
Returns true if the port is abstracted because of user-specified settings (set_port_abstraction -ignore) in the HyperScale model. This attribute applies only to ports at the block level and corresponding pins at the top level. This attribute is available only update_timing at the block level. Ports with is_user_abstracted == true also have is_abstracted == true.

is_user_dont_override

Type: boolean
Returns true if context override is not applied because of user-specified settings. This attribute applies only to ports at the block level.

launch_clocks

Type: collection
Returns a collection of the clocks that launch signals reaching the port.

max_capacitance

Type: float
Returns a floating-point number that sets the maximum capacitance value for input, output, or bidirectional ports, and designs. The units must be consistent with those of the logic library used during optimization. Set with the set_max_capacitance command.

max_fall_delay

Type: float
Returns the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_max_delay command.

max_fall_local_slack

Type: float
For a port of a transparent latch, the local slack is the timing slack considering the data arrival at the local port and setup constraints at the local port, without considering constraints downstream from the port. If there are no constraints at the port, the attribute is undefined. The normal slack (max_fall_slack or max_slack) considers the constraints downstream from the port.

max_fall_slack

Type: float
Returns the worst slack at a port for falling maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing has been updated. You cannot set this attribute.

max_fanout

Type: float
Returns the maximum fanout load for the net connected to this port. PrimeTime ensures that the fanout load on this port is less than the specified value. Set with the set_max_fanout command

max_rise_delay

Type: float
Returns the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_max_delay command.

max_rise_local_slack

Type: float
For a port of a transparent latch, the local slack is the timing slack considering the data arrival at the local port and setup constraints at the local port, without considering constraints downstream from the port. If there are no constraints at the port, the attribute is undefined. The normal slack (max_rise_slack or max_slack) considers the constraints downstream from the port.

max_rise_slack

Type: float
Returns the worst slack at a port for rising maximum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated. You cannot set this attribute.

max_slack

Type: float
Returns the minimum of the max_rise_slack and max_fall_slack attributes.

max_transition

Type: float
Returns the maximum transition time for the net connected to this port. The compile command ensures that value. Set with the set_max_transition command.

min_capacitance

Type: float
Returns the minimum capacitance value for input and bidirectional ports. The units must be consistent with those of the logic library used. Set with the set_min_capacitance command.

min_fall_delay

Type: float
Returns the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_min_delay command.

min_fall_slack

Type: float
Returns the worst slack at a port for falling minimum path delays. This attribute is valid for path

endpoints (register data pins and primary outputs) after timing is updated. You cannot set this attribute.

min_fanout

Type: float
Returns the minimum fanout design rule limit for a port.

min_rise_delay

Type: float
Returns the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with the set_min_delay command.

min_rise_slack

Type: float
Returns the worst slack at a port for rising minimum path delays. This attribute is valid for path endpoints (register data pins and primary outputs) after timing is updated. You cannot set this attribute.

min_slack

Type: float
Returns the minimum of the min_rise_slack and min_fall_slack attributes.

min_transition

Type: float
Returns the minimum transition time design rule limit for a port.

net

Type: collection
Returns a collection that contains the net connected to this port; this attribute is defined only if the port is connected to a net.

object_class

Type: string
Returns the class of the object, which is "port". You cannot set this attribute.

pg_level

Type: string
Returns PG connectivity status of the port. A return value '1' means the port has a connection to a power PG net or logic constant 1, whereas a return value '0' means the port has a connection to a ground PG net or logic constant 0.

pg_pin_info

Type: collection
Returns a collection of pg_pin_info objects with these attributes: pin_name, type, voltage_for_max_delay, voltage_for_min_delay, supply_connection.

pin_capacitance_max

Type: float
Returns the pin capacitance of a port for maximum conditions (wire capacitance is not included). Set with the set_load command.

pin_capacitance_max_fall

Type: float
Returns the maximum fall capacitance of the port. This attribute is read-only; you cannot change the setting.

pin_capacitance_max_rise

Type: float
Returns the maximum rise capacitance of the port. This attribute is read-only; you cannot change the setting.

pin_capacitance_min

Type: float
Returns the pin capacitance of a port for minimum conditions (wire capacitance is not included). Set with the set_load command.

pin_capacitance_min_fall

Type: float
Returns the minimum fall capacitance of the port. This attribute is read-only; you cannot change the setting.

pin_capacitance_min_rise

Type: float
Returns the minimum rise capacitance of the port. This attribute is read-only; you cannot change the setting.

port_direction

Type: string
Returns the direction of a port. Value can be in, out, or inout. This attribute exists for backward compatibility with dc_shell. See the "direction" attribute. You cannot set this attribute.

power_base_clock

Type: string
Returns the name of the base clock associated with this port. If the port belongs to multiple clock domains, the power_base_clock attribute is set to the fastest of the clocks.

power_rail_voltage_max

Type: float
Returns the voltage value on port or pin object for maximum condition.

power_rail_voltage_min

Type: float
Returns the voltage value on port or pin object for minimum condition.

propagated_clock

Type: boolean
Returns true if clock edge times are delayed by propagating the values through the clock network. Affects all sequential cells in the transitive fanout of this port. If this attribute is not present, PrimeTime assumes ideal clocking. Set with set_propagated_clock command.

receiver_model_scaling_libs_max

Type: collection
Returns a collection of library objects used for receiver model scaling, where applicable, for libs maximum analysis.

receiver_model_scaling_libs_min

Type: collection
Returns a collection of library objects used for receiver model scaling, where applicable, for libs minimum analysis.

receiver_model_type_max_fall

Type: string
Returns the receiver model type, either basic (NLDM) or advanced (CCS timing), for type maximum fall analysis.

receiver_model_type_max_rise

Type: string
Returns the receiver model type, either basic (NLDM) or advanced (CCS timing), for type maximum rise analysis.

receiver_model_type_min_fall

Type: string
Returns the receiver model type, either basic (NLDM) or advanced (CCS timing), for type minimum fall analysis.

receiver_model_type_min_rise

Type: string
Returns the receiver model type, either basic (NLDM) or advanced (CCS timing), for type minimum rise analysis.

setup_uncertainty

Type: float
Returns the clock uncertainty (skew) of a clock used for setup (and other maximum delay) timing checks. Set with the set_clock_uncertainty command.

si_noise_active_aggressors_above_high

Type: collection
Returns a collection of active aggressor nets for the input port, considering crosstalk noise bumps in the above-high region. An active aggressor is an aggressor that contributes to the worst-case noise bump on the victim net.

si_noise_active_aggressors_above_low

Type: collection
Returns a collection of active aggressor nets for the input port, considering crosstalk noise bumps in the above-low region. An active aggressor is an aggressor that contributes to the worst-case noise bump on the victim net.

si_noise_active_aggressors_below_high

Type: collection
Returns a collection of active aggressor nets for the input port, considering crosstalk noise bumps in the below-high region. An active aggressor is an aggressor that contributes to the worst-case noise bump on the victim net.

si_noise_active_aggressors_below_low

Type: collection
Returns a collection of active aggressor nets for the input port, considering crosstalk noise bumps in the below-low region. An active aggressor is an aggressor that contributes to the worst-case noise bump on the victim net.

si_noise_bumps_above_high

Type: string
Returns a string that lists the aggressor nets for the input port and their corresponding coupled bump heights and widths, considering noise bumps in the above-high region. The format of the string is:

```
{{aggr1_name height width}
 {aggr2_name height width}
  ... }
```

Height is in volts and width is in library time units.

si_noise_bumps_above_low

Type: string
Returns a string that lists the aggressor nets for the input port and their corresponding coupled bump heights and widths, considering noise bumps in the above-low region. The format of the string is:

```
{{aggr1_name height width}
 {aggr2_name height width}
  ... }
```

Height is in volts and width is in library time units.

si_noise_bumps_below_high

Type: string

Returns a string that lists the aggressor nets for the input port and their corresponding coupled bump heights and widths, considering noise bumps in the below-high region. The format of the string is:

```
{{aggr1_name height width}
 {aggr2_name height width}
  ... }
```

Height is in volts and width is in library time units.

si_noise_bumps_below_low

Type: string
Returns a string that lists the aggressor nets for the input port and their corresponding coupled bump heights and widths, considering noise bumps in the below-low region. The format of the string is:

```
{{aggr1_name height width}
 {aggr2_name height width}
  ... }
```

Height is in volts and width is in library time units.

si_noise_height_factor_above_high

Type: float
Returns the noise height derating factor for the port, in the above-high region, as set by the **set_noise_derate** command.

si_noise_height_factor_above_low

Type: float
Returns the noise height derating factor for the port, in the above-low region.

si_noise_height_factor_below_high

Type: float
Returns the noise height derating factor for the port, in the below-high region.

si_noise_height_factor_below_low

Type: float
Returns the noise height derating factor for the port, in the below-low region.

si_noise_height_offset_above_high

Type: float
Returns the noise height offset for the port, in the above-high region, as set by the **set_noise_derate** command.

si_noise_height_offset_above_low

Type: float
Returns the noise height offset for the port, in the above-low region.

si_noise_height_offset_below_high

Type: float
Returns the noise height offset for the port, in the below-high region.

si_noise_height_offset_below_low

Type: float
Returns the noise height offset for the port, in the below-low region.

si_noise_prop_bumps_above_high

Type: string
Returns a string that shows the bump height and width at the input port caused by noise propagation, in the above-high region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_prop_bumps_above_low

Type: string
Returns a string that shows the bump height and width at the input port caused by noise propagation, in the above-low region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_prop_bumps_below_high

Type: string
Returns a string that shows the bump height and width at the input port caused by noise propagation, in the below-high region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_prop_bumps_below_low

Type: string
Returns a string that shows the bump height and width at the input port caused by noise propagation, in the below-low region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_slack_above_high

Type: float
Returns the amount of noise slack for the port in the above-high region.

si_noise_slack_above_low

Type: float
Returns the amount of noise slack for the port in the above-low region.

si_noise_slack_below_high

Type: float
Returns the amount of noise slack for the port in the below-high region.

si_noise_slack_below_low

Type: float
Returns the amount of noise slack for the port in the below-low region.

si_noise_total_bump_above_high

Type: string
Returns a string that shows the total bump height and width at the input port caused by crosstalk and noise propagation, in the above-high region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_total_bump_above_low

Type: string
Returns a string that shows the total bump height and width at the input port caused by crosstalk and noise propagation, in the above-low region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_total_bump_below_high

Type: string
Returns a string that shows the total bump height and width at the input port caused by crosstalk and noise propagation, in the below-high region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_total_bump_below_low

Type: string
Returns a string that shows the total bump height and width at the input port caused by crosstalk and noise propagation, in the below-low region. The format of the string is: {*height width*}. Height is in volts and width is in library time units.

si_noise_width_factor_above_high

Type: float
Returns the noise width derating factor for the port, in the above-high region.

si_noise_width_factor_above_low

Type: float
Returns the noise width derating factor for the port, in the above-low region.

si_noise_width_factor_below_high

Type: float
Returns the noise width derating factor for the port, in the below-high region.

si_noise_width_factor_below_low

Type: float
Returns the noise width derating factor for the port, in the below-low region.

si_noise_worst_prop_arc_above_high

Type: collection
Returns the cell arc that corresponds to the worst noise propagation to an input port with a driving cell.

si_noise_worst_prop_arc_above_low

Type: collection
Returns the cell arc that corresponds to the worst noise propagation to an input port with a driving cell.

si_noise_worst_prop_arc_below_high

Type: collection
Returns the cell arc that corresponds to the worst noise propagation to an input port with a driving cell.

si_noise_worst_prop_arc_below_low

Type: collection
Returns the cell arc that corresponds to the worst noise propagation to an input port with a driving cell.

temperature_max

Type: float
Returns the maximum temperature specified for the cell through the operating condition specification or application of the set_temperature command.

temperature_min

Type: float
Returns the minimum temperature specified for the cell through the operating condition specification or application of the set_temperature command.

user_case_value

Type: string
Returns the user-specified logic value of a pin or port.

wire_capacitance_max

Type: float
Returns the wire capacitance of the port for maximum conditions (pin capacitance is not included). Set with the set_load command.

wire_capacitance_max_fall

Type: float
Returns the maximum fall wire capacitance of the net. This attribute is read-only; you cannot change the setting.

wire_capacitance_max_rise

Type: float
Returns the maximum rise wire capacitance of the net. This attribute is read-only; you cannot change the setting.

wire_capacitance_min

Type: float
Returns the wire capacitance of the port for minimum conditions (pin capacitance is not included). Set with the set_load command.

wire_capacitance_min_fall

Type: float
Returns the minimum fall wire capacitance of the net. This attribute is read-only; you cannot change the setting.

wire_capacitance_min_rise

Type: float
Returns the minimum rise wire capacitance of the net. This attribute is read-only; you cannot change the setting.

wire_load_model_max

Type: string
Returns the name of a wire load model (for maximum conditions) that can be used for prelayout wire load estimation of the net connected to a port. Set with the set_wire_load_model command.

wire_load_model_min

Type: string
Returns the name of a wire load model (for minimum conditions) that can be used for prelayout wire load estimation of the net connected to a port. Set with the set_wire_load_model command.

x_coordinate

Type: float
Returns the x-coordinate of the port.

y_coordinate

Type: float
Returns the y-coordinate of the port.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

port_search_in_current_instance

Controls whether the **get_ports** command can get ports of the current instance.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), the **get_ports** command gets ports of the current design. If this variable is set to **true**, the **get_ports** command gets ports of the current instance.

SEE ALSO

`get_ports(2)`

power_activity_file_precedence_over_sca

Enables or disables activity annotation to take precendence over constants (logic constants or case contants) in PrimePower. This variable works only in average analysis mode.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When set to **false** , constants take precedence.

power_activity_precedence_over_force_implied

Enables or disables activity annotation to take precendence over "set_switching_activity -force" propagated nets in PrimePower. The activity type for "set_switching_activity -force" propagated nets is "force_implied". By default, file and SSA annotation have higher precedence than "force_implied". This variable works in both average and time based analysis mode.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When set to **false** , "force_implied" take precedence over file and set_switching_activity.

power_analysis_mode

Sets the power analysis mode.

TYPE

string

DEFAULT

averaged

DESCRIPTION

This variable explicitly selects the analysis mode for power calculation. PrimePower provides three different analysis modes: **averaged**, **time_based**. Set this variable before the first power command, otherwise, the default mode is assumed. For a particular analysis mode, you must provide the appropriate activity information. The allowed values are as follows:

- **averaged** (the default): PrimePower calculates power based on toggle-rate and state-probability. Only averaged power results are calculated. In this mode, it can take the VCD file and SAIF file as activity input files. Use the **set_switching_activity** and **set_case_analysis** commands to set the statistical switching activity on top of the default switching activity. For more information, see the **update_power** man page.
- **time_based**: PrimePower calculates power based on the events from VCD. Averaged power results are calculated, along with calculations for peak powers and time-based power waveforms. You must provide the VCD file in this mode. Both gate-level VCD and RTL VCD can be specified for this mode. For more information, see the **update_power** man page.

  The **power_analysis_mode** variable can change in one run. However, if you change the setting, all the activity and power data is removed internally. You must provide activity information before the **update_power** command.

  In addition, you can use the **set_power_analysis_options** to specify the options for power

analysis.

SEE ALSO

read_saif(2)
read_vcd(2)
report_power(2)
set_case_analysis(2)
set_power_analysis_options(2)
set_switching_activity(2)
update_power(2)
power_enable_analysis(3)

power_calc_use_ceff_for_internal_power

Specifies whether to use effective C for internal power calculation.

TYPE

Boolean

DEFAULT

false

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.

power_capp_edge_triggered_propagation

Controls to perform event propagation through edge triggered sequential cells in CAPP flow.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

In CAPP flow, all sequential outputs are expected to be annotated, which doesn't happen always in practice. In such cases, PrimePower supports propagation through sequential elements, however, propagation through edge triggered sequential elements sometimes lead to overestimation of power. This variable controls enables/disables propagation through edge triggered sequential elements in CAPP flow. By default, propagation through edge triggered elements is enabled in PrimePower.

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 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 the **-include** or **-exclude** options. The default check list defined by this variable can be overridden by either redefining it before the **check_power** command 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 is issued by the next execution of the **check_power** command.

SEE ALSO

`check_power(2)`

power_clock_network_include_clock_gating_ne

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 set to **true**, discrete logic structure that functions as clock gating belongs to the **clock_network** power group. Only the typical clock gating logic is considered a qualified clock gating network included in the clock network. The typical clock gating logic starts from the output of a level-sensitive latch driven by the specified clock. The clock gating logic possibly goes through some buffers, and returns 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 enabled 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 part of the **clock_network** power group, but not the **register** power group.

SEE ALSO

```
create_power_group(2)
report_power(2)
```

power_clock_network_include_register_clock_p

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 set to its default of **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 set to **false**, the power is included as **register** power and excluded from **clock_network** power.

SEE ALSO

`create_power_group(2)`
`report_power(2)`

power_default_base_clock

Specifies the default power_base_clock to be be used by PrimePower.

TYPE

String

DEFAULT

fastest

DESCRIPTION

This variable allows PrimePower to use the specified clock as the default power_base_clock . For unconstrained pins/nets , usually the fastest clock in the design is selected as the power_base_clock. This variable can be used to overrirde this default behaviour.

This should be specified after design contraints (SDC) are read and prior to set_switching_activity/update_power command.

SEE ALSO

`set_power_base_clock(2)`

power_default_static_probability

Specifies the default static probability value.

TYPE

float

DEFAULT

0.5

DESCRIPTION

The **power_default_static_probability** variable (along with the **power_default_toggle_rate** and **power_default_toggle_rate_reference_clock** variables) determines the switching activity of unannotated nets that are driven by primary inputs or black box cells. The **power_default_toggle_rate** variable specifies the default toggle rate value. The **power_default_toggle_rate_reference_clock** variable specifies how the related clock for default toggle rate is determined.

For other nonannotated nets, PrimePower 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:

- Annotated values are used.
- In some cases, unannotated switching activity values can still be accurately derived. For example, if the net drives a buffer cell and the output of this cell is annotated, your annotated values are used as the default values. Also, if the input is a clock the clock period and waveform are used to derive the switching activity values.
- If the static probability is not annotated, the value of the **power_default_static_probability** variable is used for the static probability value.

- If the toggle rate is not annotated by you, whether 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 **power_default_toggle_rate_reference_clock** variable determines the related clock. You can set this variable to these settings:

- **fastest** - The related clock is the fastest clock in the design.
- **related** - The related clock depends on the clock domain that 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, the value of **power_default_toggle_rate** should be 0.0. If the value of **power_default_toggle_rate** is 0.0, the value of the **power_default_static_probability** should be either 0.0 or 1.0.

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

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**, **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 nonannotated nets, PrimePower 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 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 of **power_default_static_probability** is 0.5 and the default of **power_default_toggle_rate** is 0.1.

SEE ALSO

set_switching_activity(2)
power_default_static_probability(3)
power_default_toggle_rate_reference_clock(3)

power_default_toggle_rate_reference_clock

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

TYPE

string

DEFAULT

related

DESCRIPTION

The **power_default_toggle_rate_reference_clock**, **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 nonannotated nets, PrimePower 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, the following is used for the toggle rate value regardless of whether the static probability is set or not:

  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 of both variables is 0.5.

SEE ALSO

```
set_switching_activity(2)
power_default_static_probability(3)
power_default_toggle_rate(3)
```

power_disable_exact_name_match_to_hier_pin

Enables or disables exact matching of VCD or SAIF objects to hierarchical pin names in the gate-level netlist when annotating activity for PrimePower power analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, the tool does not perform exact name matching of VCD or SAIF objects to gate-level hierarchical pins. The tool honors only hierarchical pins that are explicitly mapped with the **set_rtl_to_gate_name** command

SEE ALSO

`set_rtl_to_gate_name(2)`

power_disable_exact_name_match_to_net

Disables exact name matching of the RTL VCD or SAIF objects to the net names in the gate-level netlist when annotating switching activity during power analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set the **power_disable_exact_name_match_to_net** variable to **true**, the exact name matching of the RTL VCD or SAIF objects to gate-level nets stops. Only the nets which are explicitly mapped using the *set_rtl_to_gate_name* command will be honored.

SEE ALSO

`set_rtl_to_gate_name(2)`

power_domains_compatibility

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

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **true**, PrimeTime reverts to power domains mode and disables 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 is in UPF mode by default, and all power domain commands are not available.

If 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**.

If you change this variable after library loading, you must reload the libraries because the tool performs library PG conversion and update in UPF mode but not in power domains mode. For more information, see the man page for the **library_pg_file_pattern** variable.

SEE ALSO

library_pg_file_pattern(3)

power_em_disable_extrapolation

Enables or disables extrapolation for PrimePower electromigration analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, PrimePower will extrapolate one extra grid point outside EM table. If you set this variable to **true**, the tool will stop extrapolating outside table range and will cap the value at boundary points.

power_enable_advanced_fsdb_reader

Enables or disables advanced fsdb reader for PrimePower.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, enables advanced fsdb reader for PrimePower to support regular fsdb file reading as well as virtual fsdb file reading. Without this variable set to **true**, virtual fsdb file reading is not supported. By default, **power_enable_advanced_fsdb_reader** is disabled for PrimePower.

power_enable_analysis

Enables or disables PrimePower, which provides power analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, enables PrimePower, so that you can perform power analysis. Without this variable set to **true**, you cannot see power related data. By default, PrimePower is disabled; this variable is set to **false**.

If you set this variable to **true** and enable PrimePower, you must obtain a PrimePower license. You cannot use PrimePower without a license.

SEE ALSO

`report_power(2)`

power_enable_clock_scaling

Enables or disables clock scaling for power analysis in PrimePower.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, this variable enables PrimePower to scale average power number according to clock frequencies specified in SDC and the **set_power_clock_scaling** command.

SEE ALSO

set_power_clock_scaling(2)

power_enable_concurrent_event_analysis

Enables or disables concurrent event analysis in time based mode in PrimePower.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, enables concurrent event analysis in time-based mode in PrimePower. This can be useful for speeding up the power analysis runtime with a large FSDB file. It is recommended to use this variable with 'read_fsdb' command on a gate-level FSDB. With concurrent event analysis enabled , the analysis is run with multiple processes over different time windows and the power results are then merged. The time window is split by PrimePower based on the user specified time interval in 'read_fsdb' command. If no time interval is specified , then the total simulation window in the FSDB is used . PrimePower will check if the FSDB data in the given time window is large enough to perform concurrent event analysis.

SEE ALSO

`report_power(2)`

power_enable_delay_shifted_event_analysis

Enables or disables delay shifted event analysis in PrimePower.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, enables delay aware peak power analysis in PrimePower using RTL or Zero-delay VCD/FSDB . With the feature enabled , users can get realistic power waveforms with RTL or Zero-delay FSDB/VCD using the delays obtained from PrimeTime. This is expected to give an accuracy comparable to peak power analysis done using SDF annotated Gate-level FSDB/VCD. Appropriate delays are added to each events in the activity file and power is computed using these delay shifted events. For CAPP flow , delays are added to propagated events as well. For this feature to work , the power analysis mode must be set to time_based and the **"rtl"** or **"zero_delay"** option (as appropriate) must be provided when reading the VCD or FSDB file. Peak power is calculated using a sampling interval of 1ps. The "-waveform_interval" or "-cycle_accurate_clock" or "-cycle_accurate_cycle_count" options in set_power_analysis_options command are ignored in this flow.

SEE ALSO

`set_power_delay_shifted_event_analysis_options(2)`

power_enable_em_analysis

Enables or disables electromigration analysis for PrimePower.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, enables electromigration analysis for PrimePower, so that you can perform electromigration analysis. Without this variable set to **true**, you cannot run EM related commands. By default, **power_enable_em_analysis** is disabled for PrimePower.

power_enable_feedthrough_in_empty_cell

Enables PrimePower to test for feedthrough nets in empty hierarchical cells.

TYPE

boolean

DEFAULT

false

DESCRIPTION

PrimePower, by default, identifies a hierarchal cell as a black box if the cell does not contain any cell inside it. If you set this variable to **true**, PrimePower also tests presence of feedthrough nets in such cases and if such net is present, the cell is no longer treated as black-box cell.

power_enable_merged_fsdb

Resolves conflict activity on the nets in the merged FSDB file. A conflict activity for a given net is defined as activities with different transitions in same time stamp. The activity for different net segments of a net can be different across hiearchical levels. This variable works in both average and time based analysis mode.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true** , resolves conflicting activity on the net in the merged FSDB file.

power_enable_multi_rail_analysis

Enables or disables PrimePower concurrent multirail power analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, PrimePower starts concurrent multirail power analysis power updates. Under concurrent multirail power analysis mode, power data for each rails or supply nets is maintained and processed individually and concurrently. Power reports of different combinations of rail and supply net specifications can be generated without the need to update power again.

The **-rails** option of the **report_power** command requires that this variable is set to **true**.

SEE ALSO

`report_power(2)`

power_enable_pin_internal_power

Enables or disables reporting of internal_power attribute on a pin.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, internal_power attribute can be queried and reported on pin. Without this variable set to **true**, you cannot query or report internal_power attribute on pin. By default, **power_enable_pin_internal_power** is disabled for PrimePower. You have to set **power_enable_pin_internal_power** before **update_power**.

SEE ALSO

```
report_attribute(2)
get_attribute(2)
list_attribute(2)
```

power_enable_saif_all_state_static_probability

Enables or disables computation of static probability including all possible logic states from SAIF.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, PrimePower computes the static probability taking into account all possible logic states included in SAIF - '1', '0', 'X', 'Z', and 'B' using the following equation:

SP = (T1 + 0.5*(TX+TZ+TB))/(T1+T0+TX+TZ+TB)

By default, PrimePower only considers logic state '1' and '0' from SAIF for computation of static probability using following equation SP = T1/(T1+T0)

power_estimate_power_for_unmatched_event

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

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Sometimes when an output pin toggles, PrimePower cannot find a matching table in the library based on the current state of the cell. This variable controls whether PrimePower 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.

power_full_transition_glitch_scaling

Enables the scaling of transition times used to identify and calculate the power of glitches

TYPE

float

DEFAULT

1.0

DESCRIPTION

To prevent over-estimation of glitch power, PrimePower enables scaling of the transition times used to identify and calculate the power of glitches, such that they represent the times required to reach full logic levels; instead of the times required to transition from the slew trip-points defined in the library. In the time-based analysis mode, PrimePower relies on the following formulas to identify glitches and to scale their power: Glitch Identification: If (trise + tfall) > 2 * pulse_width then pulse = glitch Glitch Power Scaling Ratio: ((2 * pulse_width)/(trise + tfall))2 The transition times derived from timing analysis represent the times to reach the trip-points (eg 20%-80%); which can cause would-be-glitches to be seen as full logic level transitions. In the K-1512 release, users can specify a scaling ratio, by which to multiply the transition times; identify more glitch transitions; and to scale their power accordingly.

This featured is enabled with the following variable setting: set power_full_transition_glitch_scaling <value> The value can be any real number between 1.0 and 10.0. The default value is 1.0

power_include_initial_x_transitions

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

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Initially, if you do not set a logic value to a certain net, PrimePower assumes the logic value is X. In the later stage, the value becomes *0* or *1*. This variable controls whether PrimePower should count the power caused by the X->0 or X->1 toggle.

power_keep_vcd_event_order

Enables PrimePower to preserve order of events read from VCD/FSDB in CAPP flow.

TYPE

boolean

DEFAULT

false

DESCRIPTION

PrimePower , by default reorders events in CAPP flow so that output events occur after input events within a time-stamp. If you disable this variable, PrimePower will preserve the order of events read from VCD/FSDB.

SEE ALSO

`report_power(2)`

power_limit_extrapolation_range

Limits extrapolation to a certain range for internal power calculations.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, PrimePower extrapolates without limits when performing internal power table lookups for out-of-range data points.

To stop extrapolation at one additional index grid when performing lookups in NLPM (nonlinear power model) internal power tables, set this variable to **true**.

power_match_state_for_logic_x

Specifies logic x interpretation.

TYPE

string

DEFAULT

0

DESCRIPTION

Specifies how PrimePower interprets logic x in the Boolean function of the **when** state of a power table. This variable 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. For example, when there is any x logic, the Boolean function is evaluated as false.

SEE ALSO

`report_power(2)`

power_order_clock_events

Enables PrimePower to process clock pin events before other data pin events read from VCD/FSDB in CAPP flow.

TYPE

boolean

DEFAULT

false

DESCRIPTION

When set to true, for all events occuring within a given time-stamp, PrimePower reorders the CLK pin events before other data pin events of a given sequential cell. It is useful when reading fsdb/vcd generated without sequence information.

SEE ALSO

`report_power(2)`

power_rail_output_file

Specifies the output file name for writing out power information for PrimeRail.

TYPE

string

DEFAULT

"" (empty)

DESCRIPTION

The **power_rail_output_file** variable is provided for PrimeRail. If the file name is provided with this variable, during power calculation relevant power information is written into the file provided. Note that information is written in a binary format. This file is then read by PrimeRail.

Starting with the B-2008.12 release, the **update_power** command can perform both average and peak power analysis. The **power_analysis_mode** variable can be used to specify the power analysis mode to perform. PrimePower can perform different types of power analysis based on the mode setting. The default power analysis mode is *averaged*. For more information, see the **power_analysis_mode** man page.

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

PrimePower can consume either time-based (for example, a VCD file) or statistical switching activity information (for example, a 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 the **read_vcd** command. The statistical switching activity can be specified by using the **read_saif** or **set_switching_activity** commands.

SEE ALSO

report_power_analysis_options(2)
set_power_analysis_options(2)
update_power(2)
power_analysis_mode(3)

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

Boolean

DEFAULT

true

DESCRIPTION

The **power_read_vcd_ignore_case** variable has been replaced with the **power_read_activity_ignore_case** variable. For backward compatibility, the **power_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.

SEE ALSO

```
read_saif(2)
read_vcd(2)
set_rtl_to_gate_name(2)
```

power_report_leakage_breakdowns

Controls to report leakage components.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

To specify whether the **report_power** command prints out leakage power components, set this variable to one of the following values:

- **false** (the default) - Does not report leakage power components. For example, only total leakage is reported.
- **true** - Reports intrinsic leakage and gate leakage in addition to the total leakage in the summary report and the cell-based power report.

SEE ALSO

`report_power(2)`

power_reset_negative_extrapolation_value

Resets the negative extrapolated energy value from library to zero.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **power_reset_negative_extrapolation_value** variable is used to reset the 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**, the negative extrapolated energy value is 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. For example, the values can 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, the derived energy number can come out negative. However, the energy number corresponding to the boundary points can be positive. Using the negative energy number given the fact that the energy number corresponding to boundary points is positive, results in incorrect energy numbers.

To resolve this problem, use the **power_reset_negative_extrapolation_value** variable, which if set to **true**, resets the negative extrapolated energy numbers to zero, if the energy number for boundary points is positive.

power_reset_negative_internal_power

Resets the negative internal power to zero.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable resets the negative internal power to zero. You can use this variable if you are confident about the accuracy of the power tables. Furthermore, if the values of capacitance and input slew values are out of range from the range specified in the power tables, the internal energy number can be negative due to extrapolation or interpolation. This variable gives you the choice to reset the negative internal energy numbers to zero.

SEE ALSO

```
report_power(2)
update_power(2)
```

power_scale_dynamic_power_at_power_off

Indicates if the dynamic power is scaled according to the static probability of the corresponding power supply net. When this variable is set to **false**, only leakage power is scaled by the power-on probability.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The statistical activity information can be input from the SAIF file, the **set_switching_activity** and **set_case_analysis** commands, or from default settings and propagated through the whole design. PrimePower 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, PrimePower 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 the power-off state, only leakage power is scaled by the power-on probability. This is the default behavior. However, if the input activity information includes toggles happened at the power-off state, both dynamic and leakage power is scaled. Under such situation, you need to set the **power_scale_dynamic_power_at_power_off** variable to **true**, so that PrimePower applies 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 averaged power analysis mode. It has no effect for time-based power analysis mode. As in time-based mode, PrimePower monitors the status of the power supply net and determines the power consumption at event basis.

SEE ALSO

power_analysis_mode(3)
read_saif(2)
set_switching_activity(2)
set_case_analysis(2)

power_scale_internal_arc

Enables scaling of state probabilities of internal power arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, this variable is set to **false**, which disables scaling. If you set this variable to **true**, the sum of the probabilities of internal arc matches with the toggle rate of the pin, even if there is no default arc for the pin in the library.

SEE ALSO

`update_power(2)`

power_table_include_switching_power

Indicates whether power tables in technology library include switch power.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Indicates whether the nonlinear power model (NLPM) in the technology library includes 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). PrimePower 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 makes the difference, especially for single cell or very small designs. For correlation and characterization verification purposes, small designs or single cell designs might 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. For example, 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. PrimePower chooses the proper power calculation approach based on this variable.

power_use_ccsp_pin_capacitance

Specifies whether to include the additional capacitance contribution of the Miller Effect when deriving capacitance values for power analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, PrimePower takes into account the waveform distortion due to the Miller Effect, when calculating the input capacitance. This setting is recommended when analyzing power for deep submicron, low voltage technologies where the gate-to-source and gate-to-drain capacitance contributions to the Miller Effect are more pronounced.

SEE ALSO

`update_power(2)`

power_x_transition_derate_factor

Sets the scale factor for X-transition power.

TYPE

float

DEFAULT

0.5

DESCRIPTION

Controls a scale factor for X-transition power.

SEE ALSO

update_power(2)

pt_ilm_dir

Specifies a directory for PrimeTime to create ILM related files.

TYPE

string

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 /dir1/dir2/ilm.

If **hier_modeling_version** is set to 2.0, the directory to create ILM related files is specified by **pt_model_dir** instead of this variable.

SEE ALSO

```
create_ilm(2)
create_si_context(2)
write_arrival_annotations(2)
hier_modeling_version(3)
pt_model_dir(3)
```

pt_model_dir

Specifies a directory in which to create model related files.

TYPE

string

DEFAULT

"." (current directory)

DESCRIPTION

Specifies a directory in which to create model related files. By default, the value is ".", the current directory. You can set this variable to any directory, such as /dir1/dir2/model.

This variable is only effective when variable **hier_modeling_version** is 2.0. All the ILM and ETM related files are in the ilm/etm subdirectory under **pt_model_dir**.

SEE ALSO

```
create_ilm(2)
create_si_context(2)
extract_model(2)
write_arrival_annotations(2)
hier_modeling_version(3)
pt_model_dir(3)
```

pt_shell_mode

Specifies the mode of operation of the current shell.

TYPE

string

DEFAULT

none

DESCRIPTION

This read-only variable specifies the mode of operation of the current shell:

- **primetime** - Current PrimeTime shell was launched in non-multi-scenario mode.
- **primetime_master** - Current shell was launched with the **-multi_scenario** option.
- **primetime_slave** - Current shell was launched by the **start_hosts** command in multi-scenario mode.

The **pt_shell_mode** variable is useful in scripts or setup files that are sourced by both the master and the slave.

SEE ALSO

`pt_shell(1)`

pt_tmp_dir

Specifies a directory for PrimeTime to use as temporary storage.

TYPE

string

DEFAULT

/tmp (the local /tmp partition)

DESCRIPTION

This variable 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.

A very important use of the disk storage specified by **pt_tmp_dir** is for the high capacity mode (for more details about high capacity mode, see **set_program_options**). From a storage point of view, there are temporary files created during the run of PrimeTime to store some data at runtime, the files stored in pt_tmp_dir are automatically removed either during the run or at the exit of the program. To monitor and identify the subdirectories and files that are still in active use inside **pt_tmp_dir**, the tool writes special empty files with names in the form of LOCK#hostname#pid to indicate the host and pid of the PrimeTime process that is actively using the files in the directory.

SEE ALSO

`set_program_options(2)`

ptxr_root

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

TYPE

string

DEFAULT

By default, this variable is the same as the root path where PrimeTime 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 the PrimeTime 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 guaranteed 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 uses the **ptxr_root** variable 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...
```

. . .

Note: When this variable is set, all downstream file reading that requires ptxr will use the reader from the specified path unless you explicitly set it to the default.

SEE ALSO

```
read_ddc(2)
read_lib(2)
read_vhdl(2)
ptxr_setup_file(3)
```

query_objects_format

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable sets the format that the **query_objects** command uses to print its result. There are two supported formats: Legacy and Tcl.

The Legacy format looks like this:

```
{"or1", "or2", "or3"}
```

The Tcl format looks like this:

```
{or1 or2 or3}
```

Please see the man page for **query_objects** for complete details.

SEE ALSO

query_objects(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 this variable is set to **true** (the default), the tool 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 **rc_rd_less_than_rnet_threshold** variable (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, the tool 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.

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 use the pin capacitances from the minimum or maximum library during minimum RC delay calculation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

To specify which pin capacitances are used during minimum RC delay calculation, set the **rc_always_use_max_pin_cap** variable to one of these values:

- **false** (the default) - Uses the pin capacitances from the minimum library.
- **true** - Uses the pin capacitances from the maximum library.

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 can be queried via attributes on driver pins and ports with driving cells.

If you set the **rc_cache_min_max_rise_fall_ceff** variable to **true** before a timing update, the tool caches the effective capacitance (Ceff) associated with propagated driver behavior according to min/max and rise/fall characteristics.

To return the cached values, query the following driver pin or port attributes:

- **cached_ceff_max_rise**
- **cached_ceff_min_rise**
- **cached_ceff_max_fall**
- **cached_ceff_min_fall**

These cached attributes are useful for obtaining the worst-case effective capacitance for every driver in the design. The values of the cached attributes depend on the selected slew-propagation mode.

Notes:

- If the **rc_cache_min_max_rise_fall_ceff** 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.
- Other effective capacitance attributes -- that is, **effective_capacitance_min**, **effective_capacitance_max**, **ceff_params_min**, and **ceff_params_max**) -- are computed at query time; therefore returning those values takes considerably more runtime.

The following example shows 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.

SEE ALSO

remove_annotated_parasitics(2)

rc_ccs_extrapolation_range_compatibility

Enables enhanced CCS extrapolation.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Set this variable to one of these values:

- **true** (the default) - Disables the enhanced CCS extrapolation feature.
- **false** - Enables enhanced extrapolation for CCS library data when any extrapolation occurs.

SEE ALSO

```
timing_max_capacitance_derate(3)
timing_max_transition_derate(3)
RC-011(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.

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 (such as an extracted timing model), this criterion can be met without a sufficient number of C-effective iterations.

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

Note: 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.

rc_degrade_min_slew_when_rd_less_than_rne

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

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is **false** (the default), the tool does not use slew degradation through RC networks in min analysis mode during the RC-009 condition. When **true**, the tool 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 the tool 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, the tool 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, the tool 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.

SEE ALSO

```
rc_adjust_rd_when_less_than_rnet(3)
rc_filter_rd_less_than_rnet(3)
rc_rd_less_than_rnet_threshold(3)
RC-009(n)
```

rc_degrade_wlm_net_slew_based_on_delay

Calculates the net slew degradation for wire load model based on net delay if the library does not have slew degradation data.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, PrimeTime does not compute the net slew degradation for wire load model if the library does not have slew degradation data.

If you set this variable to **true**, PrimeTime calculates the net slew degradation for wire load model based on net delay if library does not have slew degradation data.

SEE ALSO

```
report_delay_calculation(2)
report_timing(2)
update_timing(2)
```

rc_driver_model_mode

Specifies the driver model type for RC delay calculation.

TYPE

string

DEFAULT

advanced

DESCRIPTION

To specify the driver model type for RC delay calculation, set the **rc_driver_model_mode** variable to one of these values:

- **basic** - Uses the driver models derived from the conventional delay and slew schema present in design libraries.
- **advanced** - Uses the advanced driver model if Composite Current Source (CCS) data is available. The **report_delay_calculation** command used on a cell arc shows the "Advanced driver modeling" message.

If the **rc_driver_model_mode** variable is set to **basic**, and the variable **rc_receiver_model_mode** is set to **advanced**, the tool uses the advanced voltage-dependent capacitance models to derive an equivalent single capacitance dependent only on the rise, fall, minimum, or maximum arc condition. These equivalent capacitances are used in analysis instead of the pin capacitances from the library.

SEE ALSO

```
report_delay_calculation(2)
rc_receiver_model_mode(3)
```

rc_filter_rd_less_than_rnet

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

TYPE

Boolean

DEFAULT

true

DESCRIPTION

To control the display of RC-009 messages, set the **rc_filter_rd_less_than_rnet** variable to one of these values:

- **true** (the default) - Displays the RC-009 message only when a network delay is greater than the corresponding driver transition time.
- **false** - 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. If the drive resistance 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, the tool 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 the time the driver reaches its later slew trip point.

In addition to the **rc_filter_rd_less_than_rnet** variable, the following variables also affect the RC-009

condition:

- **rc_adjust_rd_when_less_than_rnet**
- **rc_degrade_min_slew_when_rd_less_than_rnet**
- **rc_rd_less_than_rnet_threshold**

SEE ALSO

```
rc_adjust_rd_when_less_than_rnet(3)
rc_degrade_min_slew_when_rd_less_than_rnet(3)
rc_rd_less_than_rnet_threshold(3)
RC-009(n)
```

rc_rd_less_than_rnet_threshold

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

TYPE

float

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**.

To determine the value appropriate for a given technology, look at PrimeTime accuracy versus drive strength for cells connected to very resistive networks, such as top-level routes, with **rc_adjust_rd_when_less_than_rnet** set false. At a particular drive strength, the accuracy suffers 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 (such as, 1.0). Next, gradually decrease **rc_rd_less_than_rnet_threshold** until RC-009 switches off at the specified drive strength.

As technology shrinks, so do drive resistances, causing an increased occurrence of RC-009. In that case, you can decrease the **rc_rd_less_than_rnet_threshold** variable or switch to using Composite Current Source (CCS) data for delay calculation.

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 the receiver model type for RC delay calculation.

TYPE

string

DEFAULT

advanced

DESCRIPTION

To specify the receiver model type for RC delay calculation, set the **rc_receiver_model_mode** variable to one of these values:

- **basic** - Uses the pin capacitances specified in the design libraries. The basic model is a single capacitance dependent only on the rise, fall, minimum, or maximum arc condition.
- **advanced** - Uses the advanced receiver model if data for it is present, and if the network is driven by the advanced driver model. The advanced model is a voltage-dependent capacitance additionally dependent on input-slew and output capacitance. The advanced receiver model is part of the Synopsys Composite Current-Source (CCS) model. The **report_delay_calculation** command used on a network arc shows the "Advanced receiver modeling" message.

The advanced model has many advantages, such as the improvement accuracy of delays and slews. Another advantage is that nonlinearities, such as the Miller effect, are addressed.

When the **rc_receiver_model_mode** variable is set to **advanced**, and the network is not driven by the advanced driver model -- for example, the variable **rc_driver_model_mode** is set to **basic** or lumped load is used -- the tool uses the advanced voltage-dependent capacitance models to derive an equivalent single capacitance dependent only on the rise, fall, minimum, or maximum arc condition. These equivalent capacitances are used in analysis instead of the pin capacitances from the library. The **report_delay_calculation** command used on a network arc does not show the "Advanced receiver

modeling" message for these calculations, since only an equivalent single capacitance is used.

SEE ALSO

report_delay_calculation(2)
rc_driver_model_mode(3)

read_parasitics_load_locations

Specifies that **read_parasitics** should load location information during the reading of a parasitics file.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, **read_parasitics** loads the locations of various nodes of nets, pins, and ports that are present in the parasitic file. The default is **false**, in which case location data is 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 are also considered for the net's bounding box.

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

SEE ALSO

`read_parasitics(2)`

report_capacitance_use_ccs_receiver_model

Specifies whether the basic or advanced receiver model is used to report receiver pin capacitance.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true**, the advanced CCS receiver model is used to report receiver pin capacitance by the **report_net**, **report_attribute**, **report_delay_calculation**, and **report_timing** commands. When it is set to **false**, the basic library-derived lumped pin capacitance is used. The variable setting only affects pin capacitance reporting, not delay calculation.

SEE ALSO

```
report_timing(2)
report_net(2)
report_delay_calculation(2)
```

report_default_significant_digits

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

TYPE

integer

DEFAULT

2

DESCRIPTION

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.

SEE ALSO

`report_timing(2)`

scaling_calculation_compatibility

Enables the usage of 2016.06 and earlier scaling algorithms which is less accurate.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Set this variable to one of these values:

- **true** (the default) - Disables the more accurate new library scaling, use the old scaling method of 2016.06 and earlier releases.
- **false** - Enables new nonlinear scaling optimization to achieve better accuracy, particularly for low voltage regions. Also enables a better handling of the mismatched slew index range across different libraries, this could reduce the number of RC-011s due to mismatched slew index ranges.

SEE ALSO

`define_scaling_lib_group(2)`
`RC-011(n)`

scenario_attributes

Describes the predefined application attributes for scenario objects.

DESCRIPTION

name

Type: string
Returns the name of the scenario object.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

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

Boolean

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 following timing exceptions commands:

- **set_false_path**
- **set_multicycle_path**
- **set_max_delay**
- **set_min_delay**

Note: You can change the setting of this variable only if no exceptions have been input. If at least one exception command has already been successfully input, attempting to set this variable results in the CMD-013 error message being issued, and the variable value remains unchanged.

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 accurately tracked.

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

EXAMPLES

Here is an example of the timing exceptions report using the **report_exceptions** command.

```
pt_shell> report_exceptions

****************************************
Report : exceptions
 ...
****************************************

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 ]
```

Here is another example of the timing exceptions report using the **report_timing -exceptions** command.

```
pt_shell> report_timing -exceptions all

****************************************
Report : timing
        -path_type full
        -delay_type max
        -max_paths 1
        -exceptions all
 ...
****************************************

[ 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

Specifies the Synopsys Design Constraints (SDC) version that was written.

TYPE

string

DEFAULT

2.0

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

Specifies whether ambiguous hierarchical names are made unambiguous when they are written to SDC files.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If this variable is set to **true** (the default), the application ensures that cell, net, pin, lib_cell, and lib_pin names written to the Synopsys Design Constraints (SDC) file are not ambiguous. 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 the **-hsc** option of the following SDC object access commands:

```
get_cells
get_lib_cells
get_lib_pins
get_nets
get_pins
```

By default, PrimeTime and Design Compiler writes 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 **sdc_write_unambiguous_names** variable to **false**. The **write_sdc** command issues a warning if you set this variable to **false**.

SEE ALSO

`write_sdc(2)`

sdf_align_multi_drive_cell_arcs

Specifies whether PrimeTime unifies 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. If you set the **sdf_align_multi_drive_cell_arcs** variable to **true**, PrimeTime attempts to align the delays between the driver pins of the parallel network and the load pins of the network. The cell and net delay arcs written to the SDF file is adjusted to make this happen. The net arcs are only altered if you set the **sdf_enable_port_construct** variable to **true**. Therefore, to eliminate the small timing differences, you must set both the **sdf_align_multi_drive_cell_arcs** and **sdf_enable_port_construct** variables 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 three-state buffers.

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 multidrive cell arcs are aligned during the **write_sdf** command.

TYPE

float

DEFAULT

1

DESCRIPTION

Small timing differences in the timed switching characteristics of the mesh arcs can cause the simulation to fail. If you set the **sdf_align_multi_drive_cell_arcs** variable to **true**, PrimeTime attempts to unify the delays between the driver pins of the parallel network and the load pins of the network. The cell delay arcs written to the Standard Delay Format (SDF) file is adjusted to make this happen. The criteria for this to occur is that the delay values of the parallel cells are within a threshold of each other, where the threshold is specified by the **sdf_align_multi_drive_cell_arcs_threshold** variable. The threshold value is specified in pico seconds (ps), where the default value is 1 ps.

SEE ALSO

```
sdf_align_multi_drive_cell_arcs_threshold(3)
sdf_enable_port_construct(3)
sdf_enable_port_construct_threshold(3)
```

sdf_enable_cond_start_end

Enables or disables support for the **sdf_cond_start** and **sdf_cond_end** attributes.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, PrimeTime supports the **sdf_cond_start** and **sdf_cond_end** attributes on timing arcs. These attributes impact the way the **read_sdf** and **write_sdf** commands deal with timing arcs.

SEE ALSO

`read_sdf(2)`

sdf_enable_port_construct

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

TYPE

Boolean

DEFAULT

false

DESCRIPTION

For designs with high-fanin, high-fanout mesh clock networks, large Standard Delay Format (SDF) files are produced. If you set the **sdf_enable_port_construct** variable to **true**, PrimeTime attempts to reduce the size of the produced SDF file. PrimeTime uses the port construct instead of the interconnect construct when executing the **write_sdf** command.

Using the port construct is restricted by the **sdf_enable_port_construct_threshold** variable. Any group of parallel nets in the design that are not driven or driving three-state buffers and that have a delay value within the threshold as defined by the **sdf_enable_port_construct_threshold** variable is written out using a port construct. Otherwise, the interconnect construct is used. The port construct is not 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 is written out first, followed by interconnect statements.

SEE ALSO

`sdf_enable_port_construct_threshold(3)`

sdf_enable_port_construct_threshold

Sets the threshold value for the parallel net arcs delay variance below which parallel nets are written out using the port construct during the **write_sdf** command.

TYPE

float

DEFAULT

1

DESCRIPTION

For designs with high-fanin, high-fanout mesh clock networks, large Standard Delay Format (SDF) files are produced. If you set the **sdf_enable_port_construct** variable to **true**, PrimeTime attempts 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 are written out using the port construct.

The threshold value is specified in picoseconds (ps).

SEE ALSO

`sdf_enable_port_construct(3)`

search_path

Specifies a list of directories that contain design files, library files, and scripts.

TYPE

list

DEFAULT

"" (empty)

DESCRIPTION

This variable specifies a list of directories where PrimeTime searches for design files, library files, and scripts. Normally, the **search_path** variable is set to a central library directory.

The default of this variable is the empty string, "".

The **read_db** and **link_design** commands particularly depend on the **search_path** variable.

The **source** command searches for scripts using the **search_path** variable if you set the **sh_source_uses_search_path** variable to **true**.

SEE ALSO

```
link_design(2)
read_db(2)
source(2)
sh_source_uses_search_path(3)
```

sh_allow_tcl_with_set_app_var

Allows the **set_app_var** and **get_app_var** commands to work with application variables.

TYPE

string

DEFAULT

application specific

DESCRIPTION

Normally the **get_app_var** and **set_app_var** commands only work for variables that have been registered as application variables. Setting this variable to **true** allows these commands to set a Tcl global variable instead.

These commands issue a CMD-104 error message for the Tcl global variable, unless the variable name is included in the list specified by the **sh_allow_tcl_with_set_app_var_no_message_list** variable.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_allow_tcl_with_set_app_var_no_message_list(2)
```

sh_allow_tcl_with_set_app_var_no_message_l

Suppresses CMD-104 messages for variables in this list.

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable is consulted before printing the CMD-104 error message, if the **sh_allow_tcl_with_set_app_var** variable is set to **true**. All variables in this Tcl list receive no message.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_allow_tcl_with_set_app_var(2)
```

sh_arch

Indicates the system architecture of your machine.

TYPE

string

DEFAULT

platform-dependent

DESCRIPTION

The **sh_arch** variable is set by the application to indicate the system architecture of your machine. Examples of machines being used are sparcOS5, amd64, and so on. This variable is read-only.

sh_command_abbrev_mode

Sets the command abbreviation mode for interactive convenience.

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable sets the command abbreviation mode as an interactive convenience. Script files should not use any command or option abbreviation, because these files are then susceptible to command changes in subsequent versions of the application.

Although the default value is **Anywhere**, it is recommended that the site startup file for the application set this variable to **Command-Line-Only**. It is also possible to set the value to **None**, which disables abbreviations altogether.

To determine the current value of this variable, use the **get_app_var sh_command_abbrev_mode** command.

SEE ALSO

sh_command_abbrev_options(3)
get_app_var(2)
set_app_var(2)

sh_command_abbrev_options

Turns off abbreviation of command dash option names when false.

TYPE

boolean

DEFAULT

application specific

DESCRIPTION

When command abbreviation is currently off (see sh_command_abbrev_mode) then setting this variable to false will also not allow abbreviation of command dash options. This variable also impacts abbreviation of the values specified to command options that expect values to be one of an allowed list of values.

This variable exists to be backward compatible with previous tool releases which always allowed abbreviation of command dash options and option values regardless of the command abbreviation mode.

It is recommended to set the value of this variable to false.

To determine the current value of this variable, use the **get_app_var sh_command_abbrev_options** command.

SEE ALSO

sh_command_abbrev_mode(3)
get_app_var(2)
set_app_var(2)

sh_command_log_file

Specifies the name of the file to which the application logs the commands you executed during the session.

TYPE

string

DEFAULT

empty string

DESCRIPTION

This variable specifies the name of the file to which the application logs the commands you run during the session. By default, the variable is set to an empty string, indicating that the application's default command log file name is to be be used. If a file named by the default command log file name cannot be opened (for example, if it has been set to read only access), then no logging occurs during the session.

This variable can be set at any time. If the value for the log file name is invalid, the variable is not set, and the current log file persists.

To determine the current value of this variable, use the **get_app_var sh_command_log_file** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
```

sh_continue_on_error

Allows processing to continue when errors occur during script execution with the **source** command.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

This variable is deprecated. It is recommended to use the **-continue_on_error** option to the **source** command instead of this variable because that option only applies to a single script, and not the entire application session.

When set to **true**, the **sh_continue_on_error** variable allows processing to continue when errors occur. Under normal circumstances, when executing a script with the **source** command, Tcl errors (syntax and semantic) cause the execution of the script to terminate.

When **sh_continue_on_error** is set to **false**, script execution can also terminate due to new error and warning messages based on the value of the **sh_script_stop_severity** variable.

To determine the current value of the **sh_continue_on_error** variable, use the **get_app_var sh_continue_on_error** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_script_stop_severity(3)
```

sh_dev_null

Indicates the current null device.

TYPE

string

DEFAULT

platform dependent

DESCRIPTION

This variable is set by the application to indicate the current null device. For example, on UNIX machines, the variable is set to **/dev/null**. This variable is read-only.

SEE ALSO

`get_app_var(2)`

sh_enable_line_editing

Enables the command line editing capabilities in PrimeTime.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If this variable set to its default of **true**, advanced UNIX-like shell capabilities are enabled.

This variable needs to be set in the .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, the **sh_line_editing_mode** variable can be set in either the .synopsys_pt.setup file or directly in the shell.

Command Completion

The editor can complete commands, options, variables, and files given a unique abbreviation. You must type part of a word and press the Tab key to get the complete command, variable, or file. For command options, type **-**, and press the 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, all the matching commands, options, files, or variables are automatically listed.

Completion works in following context sensitive ways:

- 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 file names
- After a set, unset or printvar command : completes variables
- After '$' symbol : completes variables
- After the help command : completes command
- After the man command : completes commands or variables
- Any token which is not the first token and does not match any of the preceding rules : completes file names

SEE ALSO

```
list_key_bindings(2)
sh_line_editing_mode(3)
```

sh_enable_page_mode

Displays long reports one page at a time (similar to the UNIX **more** command.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

This variable displays long reports one page at a time (similar to the UNIX **more** command), when set to **true**. Consult the man pages for the commands that generate reports to see if they are affected by this variable.

To determine the current value of this variable, use the **get_app_var sh_enable_page_mode** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
```

sh_enable_stdout_redirect

Allows the redirect command to capture output to the Tcl stdout channel.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When set to **true**, this variable allows the redirect command to capture output sent to the Tcl stdout channel. By default, the Tcl **puts** command sends its output to the stdout channel.

SEE ALSO

get_app_var(2)
set_app_var(2)

sh_enable_system_monitoring

Specifies whether critical system resources monitoring is enabled or disabled.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If this variable is **true**, critical system resources are monitored throughout the run. When critical events occur, PrimeTime notifies you by issuing appropriate warning messages explaining the event that has arisen.

If this variable is **false**, critical system resources are not monitored.

sh_global_per_message_limit

Specifies the limit applied to all messages, other than those whose limit is set explicitly by the **set_message_info** command.

TYPE

integer

DEFAULT

10000

DESCRIPTION

This variable defines the limit for all messages, except those with a limit set using the **set_message_info** command.

This variable is distinct from the **sh_message_limit** variable. Unlike that variable, **sh_global_per_message_limit** applies to all messages, and is not limited in scope to any particular command or commands. Messages defined by the **sh_limited_messages** variable are bound by the **sh_message_limit** limit during the scope of relevant commands (such as **update_timing** and **read_parasitics**), but are additionally bound by the **sh_global_per_message_limit** limit over the course of a PrimeTime session.

To remove this limit, set the **sh_global_per_message_limit** variable to 0.

SEE ALSO

`get_message_info(2)`

```
print_message_info(2)
set_message_info(2)
sh_limited_messages(3)
sh_message_limit(3)
```

sh_help_shows_group_overview

Changes the behavior of the "help" command.

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable changes the behavior of the **help** command when no arguments are specified to help. Normally when no arguments are specified an informational message with a list of available command groups is displayed.

When this variable is set to false the command groups and the commands in each group is printed instead. This variable exists for backward compatibility.

SEE ALSO

help(2)
set_app_var(2)

sh_high_capacity_effort

Specifies the level of capacity effort for the timing analysis of PrimeTime and PrimeTime SI. It only provides simple heuristics for trade-off between capacity and performance of the program. It does not have any impact on the analysis results.

TYPE

string

DEFAULT

default

DESCRIPTION

Specifies the effort level for capacity improved mode of the program. Allowed values are **default**, **low**, **medium**, and **high**. The **default** value corresponds to the **medium** setting.

When effort level increases, the peak memory required by the tool is expected to reduce, with potentially slightly longer runtime. It should be clarified that this variable only provides simple heuristic control on the trade-off 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 using the **set_program_options -enable_high_capacity** command.

If the program is already in high capacity mode, further change of this variable do not have any effect until the next time the above command is issued.

SEE ALSO

set_program_options(2)

sh_high_capacity_enabled

Specifies whether high capacity mode is enabled.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This read-only variable specifies the mode of analysis. The value of this variable changes after you run the **set_program_options** command.

This variable and the **sh_high_capacity_effort** variable are saved and restored in the session data. For more information, see the **set_program_options** man page.

SEE ALSO

```
set_program_options(2)
sh_high_capacity_effort(3)
```

sh_launch_dir

Specifies 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 changes its current working directory multiple times but the **sh_launch_dir** variable remains constant across all slaves and the master.

sh_limited_messages

Specifies the set of message types that have a limit by default when the **read_parasitics**, **report_annotated_parasitics** with the **-check** option, **read_sdf**, or **update_timing** command is invoked. This limit is defined by the **sh_message_limit** variable.

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 PTE-114 SDF-036 UITE-494 LNK-039 LNK-038 LNK-043 LNK-044 HS-015 HS-031 PTE-101 PTIO-5 UITE-504"

DESCRIPTION

This variable defines the set of messages that have a limit by default when the **read_parasitics**, **report_annotated_parasitics** with the **-check** option, **read_sdf**, **report_constraint**, or **update_timing** command is executed. This limit has no impact on messages that are emitted from other commands.

This limit is refreshed per invoking of the **read_parasitics**, **report_annotated_parasitics** with the **-check** option, **read_sdf**, **report_constraint**, or implicit or explicit **update_timing** command. Each time one of these commands is invoked the **sh_message_limit** command allows a number of messages to show up for each message type specified with the **sh_limited_messages** variable. If the limit is exceeded for one type of message, a warning message appears showing the type of message that is suppressed.

The setting of this variable has lower priority than the **set_message_info** command. If the **set_message_info** command is already used to set the limit for a message type, this command has no impact for the default limit for that message type.

This variable is distinct from the **sh_global_per_message_limit** variable, which applies to all messages, and is not limited in scope to any particular command or commands. Any message defined by the **sh_limited_messages** variable is also bound by the **sh_global_per_message_limit** limit over the c ourse of a PrimeTime session.

If it is needed to remove this default limit, either set the **sh_limited_messages** variable to "" or set the **sh_message_limit** variable to 0.

SEE ALSO

```
get_message_info(2)
print_message_info(2)
set_message_info(2)
sh_message_limit(3)
sh_global_per_message_limit(3)
```

sh_line_editing_mode

Enables vi or Emacs editing mode in the PrimeTime shell.

TYPE

string

DEFAULT

emacs

DESCRIPTION

This variable sets the command line editor mode to **emacs** (the default) or **vi**.

Use the **list_key_bindings** command to display the current key bindings and edit mode.

You can set this variable in the .synopsys_pt.setup file or directly in the shell. The **sh_enable_line_editing** variable must be set to its default of **true**.

SEE ALSO

```
list_key_bindings(2)
sh_enable_line_editing(3)
```

sh_message_limit

Specifies the default limit of messages defined by the **sh_limited_messages** variable during the **read_parasitics**, **report_annotated_parasitics -check**, **read_sdf**, and **update_timing** commands.

TYPE

integer

DEFAULT

100

DESCRIPTION

This variable defines the default limit for messages in the **sh_limited_messages** variable printed from several commands, including **link_design**, **read_parasitics**, **report_annotated_parasitics** (with the **-check** option), **read_sdf**, and **update_timing**. This limit is not used for messages issued by other commands.

This limit is refreshed when you use the **read_parasitics**, **report_annotated_parasitics -check**, **read_sdf**, or implicit or explicit **update_timing** command. Each time you invoke one of these commands, the **sh_message_limit** variable displays the number of messages that show up for each message type set using the **sh_limited_messages** variable. If the limit is exceeded for one type of message, a warning message explains that this type of message is being suppressed.

This variable setting has lower priority than the **set_message_info** command. If you use the **set_message_info** command to specify the limit for a message type, the default limit on that message type is not effective.

This variable is distinct from the **sh_global_per_message_limit** variable, which applies to all messages, and is not limited in scope to any particular command or commands. Any message which is bound by the **sh_message_limit** limit is also bound by the **sh_global_per_message_limit** limit over the course of a PrimeTime session.

To remove this default limit, set the **sh_limited_messages** variable to "" (no value), or set the **sh_message_limit** variable to 0.

SEE ALSO

```
get_message_info(2)
print_message_info(2)
set_message_info(2)
sh_limited_messages(3)
sh_global_per_message_limit(3)
```

sh_new_variable_message

Controls a debugging feature for tracing the creation of new variables.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

The **sh_new_variable_message** variable controls a debugging feature for tracing the creation of new variables. Its primary debugging purpose is to catch the misspelling of an application-owned global variable. When set to **true**, an informational message (CMD-041) is displayed when a variable is defined for the first time at the command line. When set to **false**, no message is displayed.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. See the **set_app_var command man page for details.**

Other variables, in combination with **sh_new_variable_message**, enable tracing of new variables in scripts and Tcl procedures.

Warning: This feature has a significant negative impact on CPU performance when used with scripts and Tcl procedures. This feature should be used only when developing scripts or in interactive use. When you turn on the feature for scripts or Tcl procedures, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_new_variable_message_in_proc(3)
sh_new_variable_message_in_script(3)
```

sh_new_variable_message_in_proc

Controls a debugging feature for tracing the creation of new variables in a Tcl procedure.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **sh_new_variable_message_in_proc** variable controls a debugging feature for tracing the creation of new variables in a Tcl procedure. Its primary debugging purpose is to catch the misspelling of an application-owned global variable.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. Please see the **set_app_var** command man page for details.

Note that the **sh_new_variable_message** variable must be set to **true** for this variable to have any effect. Both variables must be set to **true** for the feature to be enabled. Enabling the feature simply enables the **print_proc_new_vars** command. In order to trace the creation of variables in a procedure, this command must be inserted into the procedure, typically as the last statement. When all of these steps have been taken, an informational message (CMD-041) is generated for new variables defined within the procedure, up to the point that the **print_proc_new_vars** commands is executed.

Warning: This feature has a significant negative impact on CPU performance. This should be used only when developing scripts or in interactive use. When you turn on the feature, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message_in_proc** command.

SEE ALSO

get_app_var(2)
print_proc_new_vars(2)
set_app_var(2)
sh_new_variable_message(3)
sh_new_variable_message_in_script(3)

sh_new_variable_message_in_script

Controls a debugging feature for tracing the creation of new variables within a sourced script.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **sh_new_variable_message_in_script** variable controls a debugging feature for tracing the creation of new variables within a sourced script. Its primary debugging purpose is to catch the misspelling of an application-owned global variable.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. See the **set_app_var** command man page for details.

Note that the **sh_new_variable_message** variable must be set to **true** for this variable to have any effect. Both variables must be set to **true** for the feature to be enabled. In that case, an informational message (CMD-041) is displayed when a variable is defined for the first time. When **sh_new_variable_message_in_script** is set to **false** (the default), no message is displayed at the time that the variable is created. When the **source** command completes, however, you see messages for any new variables that were created in the script. This is because the state of the variables is sampled before and after the **source** command. It is not because of inter-command sampling within the script. So, this is actually a more efficient method to see if new variables were created in the script.

For example, given the following script a.tcl:

```
echo "Entering script"
set a 23
echo a = $a
```

```
set b 24
echo b = $b
echo "Exiting script"
```

When **sh_new_variable_message_in_script** is **false** (the default), you see the following when you source the script:

```
prompt> source a.tcl
Entering script
a = 23
b = 24
Exiting script
Information: Defining new variable 'a'. (CMD-041)
Information: Defining new variable 'b'. (CMD-041)
prompt>
```

Alternatively, when **sh_new_variable_message_in_script** is **true**, at much greater cost, you see the following when you source the script:

```
prompt> set sh_new_variable_message_in_script true
Warning: Enabled new variable message tracing -
         Tcl scripting optimization disabled. (CMD-042)
true
prompt> source a.tcl
Entering script
Information: Defining new variable 'a'. (CMD-041)
a = 23
Information: Defining new variable 'b'. (CMD-041)
b = 24
Exiting script
prompt>
```

Warning: This feature has a significant negative impact on CPU performance. This should be used only when developing scripts or in interactive use. When you turn on the feature, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message_in_script** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_new_variable_message(3)
sh_new_variable_message_in_proc(3)
```

sh_output_log_file

Specifies the name of the file to which all application output is logged.

TYPE

string

DEFAULT

"" (empty)

DESCRIPTION

This variable specifies the name of the file to which the application logs all output information during a 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.

SEE ALSO

`sh_command_log_file(3)`

sh_product_version

Indicates the version of the application currently running.

TYPE

string

DESCRIPTION

This variable is set to the version of the application currently running. The variable is read only.

To determine the current value of this variable, use the **get_app_var sh_product_version** command.

SEE ALSO

get_app_var(2)

sh_script_stop_severity

Indicates the error message severity level that would cause a script to stop running before it completes.

TYPE

string

DEFAULT

application specific

DESCRIPTION

When a script is run with the **source** command, there are several ways to get it to stop running before it completes. One is to use the **sh_script_stop_severity** variable. This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a script to stop.
- When set to **W**, the generation of one or more warning or error messages causes a script to stop.
- When set to **none**, the generation messages does not cause the script to stop.

Note that **sh_script_stop_severity** is ignored if **sh_continue_on_error** is set to **true**.

To determine the current value of this variable, use the **get_app_var sh_script_stop_severity** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)
```

sh_source_emits_line_numbers

Indicates the error message severity level that causes an informational message to be issued, listing the script name and line number where that message occurred.

TYPE

string

DEFAULT

application specific

DESCRIPTION

When a script is executed with the **source** command, error and warning messages can be emitted from any command within the script. Using the **sh_source_emits_line_numbers** variable, you can help isolate where errors and warnings are occurring.

This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a CMD-082 informational message to be issued when the command completes, giving the name of the script and the line number of the command.
- When set to **W**, the generation of one or more warning or error messages causes a the CMD-082 message.

The setting of **sh_script_stop_severity** affects the output of the CMD-082 message. If the setting of **sh_script_stop_severity** causes a CMD-081 message, then it takes precedence over CMD-082.

To determine the current value of this variable, use the **get_app_var sh_source_emits_line_numbers** command.

SEE ALSO

get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)
sh_script_stop_severity(3)
CMD-081(n)
CMD-082(n)

sh_source_logging

Indicates if individual commands from a sourced script should be logged to the command log file.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When you source a script, the **source** command is echoed to the command log file. By default, each command in the script is logged to the command log file as a comment. You can disable this logging by setting **sh_source_logging** to **false**.

To determine the current value of this variable, use the **get_app_var sh_source_logging** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
```

sh_source_uses_search_path

Indicates if the **source** command uses the **search_path** variable to search for files.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When this variable is set to \ftrue the **source** command uses the **search_path** variable to search for files. When set to **false**, the **source** command considers its file argument literally.

To determine the current value of this variable, use the **get_app_var sh_source_uses_search_path** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
search_path(3)
```

sh_tcllib_app_dirname

Indicates the name of a directory where application-specific Tcl files are found.

TYPE

string

DESCRIPTION

The **sh_tcllib_app_dirname** variable is set by the application to indicate the directory where application-specific Tcl files and packages are found. This is a read-only variable.

SEE ALSO

`get_app_var(2)`

sh_user_man_path

Indicates a directory root where you can store man pages for display with the **man** command.

TYPE

list

DEFAULT

empty list

DESCRIPTION

The **sh_user_man_path** variable is used to indicate a directory root where you can store man pages for display with the **man** command. The directory structure must start with a directory named *man*. Below *man* are directories named *cat1*, *cat2*, *cat3*, and so on. The **man** command will look in these directories for files named *file.1*, *file.2*, and *file.3*, respectively. These are pre-formatted files. It is up to you to format the files. The **man** command effectively just types the file.

These man pages could be for your Tcl procedures. The combination of defining help for your Tcl procedures with the **define_proc_attributes** command, and keeping a manual page for the same procedures allows you to fully document your application extensions.

The **man** command will look in **sh_user_man_path** after first looking in application-defined paths. The user-defined paths are consulted only if no matches are found in the application-defined paths.

To determine the current value of this variable, use the **get_app_var sh_user_man_path** command.

SEE ALSO

```
define_proc_attributes(2)
get_app_var(2)
man(2)
set_app_var(2)
```

si_analysis_logical_correlation_mode

Enables or disables logical correlation analysis during PrimeTime SI delay or noise calculation.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If this variable is set to its default of **true**, 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.

If you set this variable 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; however, the PrimeTime SI runtime is faster.

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

To specify the aggressor alignment mode used in the CCS-based gate-level simulation engine, set this variable to one of these values:

- **lookahead** (default) - Enables the lookahead alignment feature for the CCS-based gate-level simulation engine so that PrimeTime SI finds the alignment that results in worst-case receiver output delay. Note that such an alignment might not correspond to the worst-case stage delay.
- **stage** - Disables the lookahead alignment feature.

SEE ALSO

`si_enable_analysis(3)`

si_enable_analysis

Enables or disables PrimeTime SI, which performs crosstalk analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, this variable is set to **false**, which disables PrimeTime SI.

To enable PrimeTime SI, set this variable to **true**, so that the **update_timing** and **report_timing** commands perform crosstalk-aware timing calculations. If you set this variable to **true**, you must also do the following:

1. Obtain a PrimeTime SI license. You cannot use PrimeTime SI without a license.
2. Use the **read_parasitics -keep_capacitive_coupling** command to read in the coupling parasitics for your design. PrimeTime SI is useful only if the design has coupling parasitics data.

SEE ALSO

```
read_parasitics(2)
report_timing(2)
update_timing(2)
```

si_enable_multi_input_switching_analysis

Enables or disables multi-input switching (MIS) analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to one of these values:

- **false** (default) - Disables MIS analysis.
- **true** - Enables MIS analysis, which uses the delay and slew coefficients specified by the **set_multi_input_switching_coefficient** command.

  When you set this variable to **true**, the tool checks out a PrimeTime-ADV license.

SEE ALSO

```
report_multi_input_switching_coefficient(2)
reset_multi_input_switching_coefficient(2)
set_multi_input_switching_coefficient(2)
si_enable_multi_input_switching_timing_window_filter(3)
```

si_enable_multi_input_switching_timing_windo

Enables or disables the window alignment and overlap checking for multi-input switching (MIS) analysis.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Set this variable to one of these values:

- **true** (default) - Enables window alignment and overlap checking during MIS analysis. The tool collects the arrival windows of input pins, performs overlap checking, and derives the actual coefficient factor.
- **false** - Ignores all arrival window information and applies the specified coefficient factor directly for MIS analysis.

SEE ALSO

```
report_multi_input_switching_coefficient(2)
set_multi_input_switching_coefficient(2)
si_enable_multi_input_switching_analysis(3)
```

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. PrimeTime SI uses the **si_filter_accum_aggr_noise_peak_ratio** and **si_filter_per_aggr_noise_peak_ratio** variables 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:

- The peak voltage of the voltage bump induced on the victim net divided by VCC is less than the value specified with the **si_filter_per_aggr_noise_peak_ratio** variable.
- 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 specified with the **si_filter_accum_aggr_noise_peak_ratio** variable.

SEE ALSO

```
si_enable_analysis(3)
si_filter_per_aggr_noise_peak_ratio(3)
```

si_filter_keep_all_port_aggressors

Specifies whether to filter the aggressors for a victim net that is connected to a port.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **false** (the default), the normal filtering algorithm is applied. When you set this variable to **true**, all aggressors for a port net are kept. It is recommended to set this variable to **true** before using the **extract_model** command.

The tool filters aggressor nets, regardless of the variable setting, when the net

- Does not have valid parasitics or driver
- Is constant logic or the cell is a single pin cell
- Is excluded by user-specified settings

SEE ALSO

`si_enable_analysis(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. PrimeTime SI uses the **si_filter_per_aggr_noise_peak_ratio** and **si_filter_accum_aggr_noise_peak_ratio** variables 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:

- The peak voltage of the voltage bump induced on the victim net divided by VCC is less than the value specified with the **si_filter_per_aggr_noise_peak_ratio** variable.
- 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 specified with the **si_filter_accum_aggr_noise_peak_ratio** variable.

SEE ALSO

```
si_enable_analysis(3)
si_filter_accum_aggr_noise_peak_ratio(3)
```

si_ilm_keep_si_user_excluded_aggressors

Specifies whether to include user-excluded PrimeTime SI aggressors in the interface logic model (ILM).

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to one of these values:

- **false** (default) - Excludes user-excluded PrimeTime SI aggressors from the ILM. You exclude aggressors by setting nets to be constant or by using the **set_si_delay_analysis** or **set_si_noise_analysis** command.
- **true** - Includes user-excluded PrimeTime SI aggressors in the ILM.

SEE ALSO

```
create_ilm(2)
set_si_delay_analysis(2)
set_si_noise_analysis(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. The following values are allowed for the variable:

- **disabled** (default) - Disables the composite aggressor feature.

  In **disabled** composite aggressor mode, PrimeTime SI uses its original flow with composite aggressor completely off to analyze the noise.

- **statistical** - Causes PrimeTime SI to calculate noise by using the statistical composite aggressor flow.

  PrimeTime SI aggregates the effect of multiple small aggressors into a single composite aggressor, thereby reducing the computational complexity and improving the performance. In **statistical** composite aggressor mode, PrimeTime SI reduces the pessimism for noise analysis by reducing the effect of composite aggressor.

SEE ALSO

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 only affects the **report_at_endpoint** analysis mode of the noise update.

This variable applies only to combinational circuit pins because sequential cell pins are automatically (noise) endpoints.

EXAMPLES

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.

SEE ALSO

report_noise(2)
report_noise_calculation(2)
set_noise_parameters(2)

si_noise_immunity_default_height_ratio

Specifies the noise immunity default height ratio.

TYPE

float

DEFAULT

0.4

DESCRIPTION

This variable sets a noise immunity default if a noise pin is not constrained. If the noise pin has no data to compute noise immunity value, the tool calculates a default noise immunity height by multiplying this variable and VDD of the pin.

Set this variable to a value between 0.0 and 1.0.

If this variable is set to 1.0, the pin is not constrained anymore, and no noise slack is calculated. The pins with no constraints are reported as "none" by the **check_noise** command.

SEE ALSO

```
check_noise(2)
report_noise_calculation(2)
```

si_noise_limit_propagation_ratio

Specifies the maximum amount of propagated noise if the noise height passes noise immunity.

TYPE

float

DEFAULT

0.75

DESCRIPTION

During a 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 specified by the **si_noise_limit_propagation_ratio** variable. This variable only affects the **report_at_source** analysis mode of the noise update.

Set this variable to a value between 0.0 and 1.0.

SEE ALSO

set_noise_parameters(2)
update_noise(2)

si_noise_skip_update_for_report_attribute

Controls whether to skip the noise update during the **report_attribute** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether to perform implicit noise update when reporting all attributes with the **report_attribute** command.

When this variable is set to **true**, attributes that require a noise update are reported by the **report_attribute** command only if the noise information is up-to-date.

If the variable is set to **false** (the default), reporting of noise-related attribute values in **report_attribute** triggers an implicit noise update.

SEE ALSO

report_attribute(2)
update_noise(2)

si_noise_slack_skip_disabled_arcs

Skips disabled timing arcs for noise slack calculation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable applies to noise slack computation on a pin that has all its forward timing arcs disabled. For such pins, to specify whether its disabled timing arcs are considered for noise slack calculation, set this variable to one of these values:

- **false** (default) - If the pin has all forward timing arcs disabled, this setting calculates noise slacks for all timing arcs.
- **true** - If the pin has all forward timing arcs disabled, this setting does not calculate noise slack for any timing arcs.

If a pin has at least one enabled forward arc, this variable does not affect noise slack calculation. In such a case, only the enabled arcs are used to perform noise slack calculation.

EXAMPLES

On a pin with one forward arc, if the arc is disabled by the **set_case_analysis** or **set_disable_timing**

command, no noise slack is calculated for it if the **si_noise_slack_skip_disabled_arcs** variable is set to **true**. However, if the variable is set to **false**, noise slack is calculated for this arc.

SEE ALSO

```
get_timing_arcs(2)
set_case_analysis(2)
set_disable_timing(2)
```

si_noise_update_status_level

Controls the number of progress messages displayed during the update of noise analysis.

TYPE

string

DEFAULT

none

DESCRIPTION

To control the number of progress messages displayed during the noise update process, set this variable to one of these values:

- **none** (default) - Displays no messages.
- **low** - Displays messages only at the beginning and end of the update.
- **high** - Displays messages at the beginning and end of the update, and messages for the noise calculation step show the completion percentage in steps of 10 percents.

  When you set this variable 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, the **report_noise** command) that forces a noise update.

SEE ALSO

```
    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

DESCRIPTION

If you set this variable to **true**, the crosstalk delta delays for each net is derated using the derate factors from the cell driving that net.

The relevant derating factor to be applied adheres 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 are checked for a relevant derate factor.

To see what derating factors are to be applied to the net in question, first obtain the driving cell ($driving_cell), and use the following command:

```
pt_shell> report_timing_derate [get_cells $driving_cell]
```

If you use the **report_timing** command with the **-derate** option, the tool reports the nonderated crosstalk delta delay as before. In addition, the derate column reports the net derating factor used to derate the delta-free net delay.

SEE ALSO

```
report_timing(2)
report_timing_derate(2)
set_timing_derate(2)
```

si_xtalk_composite_aggr_mode

Specifies the composite aggressor mode for crosstalk delay.

TYPE

string

DEFAULT

disabled

DESCRIPTION

To specify the composite aggressor mode for PrimeTime SI crosstalk delay analysis, set this variable to one of these values:

- **disabled** (default) - Disables the composite aggressor feature.

  In the **disabled** mode, PrimeTime SI uses its original flow with composite aggressor completely off to calculate the crosstalk delay.

- **statistical** - Causes PrimeTime SI to calculate crosstalk by using the statistical composite aggressor flow.

  The **statistical** composite aggressor mode reduces the pessimism for crosstalk delay analysis by reducing the effect of composite aggressor.

  PrimeTime SI aggregates the effect of some small aggressors (including filtered ones) into a single composite aggressor, reducing the computational complexity and improving the performance.

  Due to runtime consideration, SI logical correlation analysis, case analysis, timing window overlap analysis, etc are only performed for effective aggressors. They are not being performed for small filtered aggressors that are included in the composite aggressor.

SEE ALSO

remove_si_delay_disable_statistical(2)
report_delay_calculation(2)
report_si_delay_analysis(2)
set_si_delay_disable_statistical(2)
si_xtalk_composite_aggr_noise_peak_ratio(3)
si_xtalk_composite_aggr_quantile_high_pct(3)

si_xtalk_composite_aggr_noise_peak_ratio

Controls the composite aggressor selection for crosstalk analysis.

TYPE

float

DEFAULT

0.01

DESCRIPTION

This variable specifies the threshold value as a ratio of the crosstalk bump to VDD. Below this threshold, aggressors are selected into the composite aggressor group. The default is 0.01, which means that all aggressor nets with crosstalk-bump-to-VDD ratio less than 0.01 is selected into the composite aggressor group.

This variable 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 are selected into the composite aggressor group.

SEE ALSO

```
remove_si_delay_disable_statistical(2)
report_si_delay_analysis(2)
set_si_delay_disable_statistical(2)
si_filter_accum_aggr_noise_peak_ratio(3)
si_filter_per_aggr_noise_peak_ratio(3)
si_xtalk_composite_aggr_mode(3)
```

```
si_xtalk_composite_aggr_quantile_high_pct(3)
```

si_xtalk_composite_aggr_quantile_high_pct

Controls the composite aggressor creation for statistical analysis.

TYPE

float

DEFAULT

99.73

DESCRIPTION

This variable specifies the probability in percentage format that any given real combined bump height is less than or equal to the computed composite aggressor bump height. Given the specified probability, the resulting quantile value for the composite aggressor bump height is calculated.

The default 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 is covered by the composite aggressor bump height.

SEE ALSO

```
remove_si_delay_disable_statistical(2)
report_si_delay_analysis(2)
set_si_delay_disable_statistical(2)
si_xtalk_composite_aggr_mode(3)
si_xtalk_composite_aggr_noise_peak_ratio(3)
```

si_xtalk_delay_analysis_mode

Specifies the arrival window alignment mode used for crosstalk delta delay.

TYPE

string

DEFAULT

all_paths

DESCRIPTION

This variable specifies how the alignment between victim and aggressor nets is performed to compute crosstalk delta delay in PrimeTime SI. The allowed values are **all_paths** and **all_path_edges**.

In the **all_paths** (default) alignment mode, the tool considers the full range of all possible victim net arrival times, from the earliest minimum-delay arrival to the latest maximum-delay arrival, among all paths that pass through the victim net. Any overlap between the aggressor arrival window and the range between these two extreme victim net arrival times triggers a delta delay adjustment to the victim arrival transition.

The **all_paths** window overlap mode is simple and fast, but it is also conservative because the actual victim transitions might not overlap the aggressor window. For example:

- The aggressor window might overlap a region inside the victim window where there are no actual victim transitions; the victim transitions occur before and after (but not during) the aggressor window.
- The aggressor window might overlap the early, minimum-delay victim transitions but not the late, maximum-delay victim transitions; the tool still applies delta delays to the maximum-delay transitions, even though the aggressor window does not overlap them.
- The aggressor window might overlap the late, maximum-delay victim transitions but not the early,

minimum-delay victim transitions; the tool still applies delta delays to the minimum-delay transitions, even though the aggressor window does not overlap them.

To get a more accurate analysis at the cost of more runtime, set the **si_xtalk_delay_analysis_mode** to **all_path_edges**. The **all_path_edges** mode considers the actual early or late arrival edge times, without combining them into a continuous window. For maximum-delay analysis, it considers only latest-arrival edges of paths that pass through the victim net. Similarly, for minimum-delay analysis, it considers only earliest-arrival edges. Delta delay adjustments to victim arrival times are performed only when the aggressor arrival window overlaps one or more arrival edges of the applicable type (either latest or earliest arrivals). Thus, late-arrival overlaps with the aggressor window do not affect early-arrival delay calculations, and vice versa.

The **all_path_edges** mode is more accurate (less pessimistic) than the **all_paths** mode because it can reduce the number of cases where delta delays are applied, yet it is safe for signoff because it finds all situations where the aggressor transitions can worsen the arrival time of the transition in the victim net.

SEE ALSO

`si_enable_analysis(3)`

si_xtalk_double_switching_mode

Controls the double switching detection during the PrimeTime SI timing analysis.

TYPE

string

DEFAULT

disabled

DESCRIPTION

When this variable is set to **disabled**, double switching detection is disabled.

When you set this variable to one of the following values, PrimeTime SI checks whether crosstalk bump on the switching victim could cause the output to switch twice (and cause a pulse) instead of the desired single signal propagation:

- **clock_network** - Detects 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 nonsensitive edge could actually latch the state).
- **full_design** - Detects the potential double switching in the data path as well as clock path. 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 command, you can access this information by using the **report_si_double_switching** command or by the net attributes by using the **si_has_double_switching** and **si_double_switching_slack** commands.

The **si_has_double_switching** victim net attribute is true whenever 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 does not cause double switching the **si_double_switching_slack** attribute is "POSITIVE". If the victim net load pins does not have CCS noise model information, the attribute is reported as "INFINITY".

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 and transition time are large and fed into drivers that are strong enough to amplify this. To avoid double switching, either of them can be reduced.

SEE ALSO

```
report_si_double_switching(2)
si_enable_analysis(3)
```

si_xtalk_exit_on_max_iteration_count

Specifies the maximum number of incremental timing iterations, after which PrimeTime SI exits the analysis loop.

TYPE

integer

DEFAULT

2

DESCRIPTION

This variable specifies the maximum number of incremental timing iterations. PrimeTime SI exits the analysis loop after performing up to this number of iterations.

The default of this variable is 2, meaning that PrimeTime SI exits the analysis loop after performing two iterations. To override this default, set the variable to another integer; the minimum allowed value is 1.

SEE ALSO

`si_enable_analysis(3)`

si_xtalk_max_transition_mode

Specifies whether PrimeTime SI uses uncoupled or coupled values for maximum and minimum transition constraint checks.

TYPE

string

DEFAULT

uncoupled

DESCRIPTION

This variable specifies how PrimeTime SI computes the transition for crosstalk delay analysis. The allowed values are

- **uncoupled** (the default) - Uses uncoupled transition values.
- **reliability** - Uses coupled transition values based on the delta slew.

SEE ALSO

`report_constraint(2)`

si_xtalk_use_physical_exclusivity_on_ignore_a

Performs or discards physical exclusivity analysis during PrimeTime SI delay analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to its default of **false**, PrimeTime SI discards physical exclusivity analysis while performing crosstalk delay for nets with ignore_arrival marking based on set_si_delay_analysis. For instance, consider two clocks that are physically exclusive, CLK1 and CLK2. For a victim stage, an aggressor (name 'A') clocked by CLK1 and another aggressor (name 'B') clocked by CLK2. And these aggressor nets have ignore_arrival marking based on set_si_delay_analysis. Aggressor 'A' and 'B' can simulaneously attack the victim stage. This setting is runtime efficient but can possibly result in some pessimism.

If you set this variable to **true**, the victim stage is analyzed multiple times to consider each possible victim and aggressor combination across the physically-exclusive clocks. In this case, aggressors 'A' and 'B' cannot simultaneously attack the victim stage. This removes the pessimism when nets are marked for ignore_arrival using set_si_delay_analysis.

SEE ALSO

`set_si_delay_analysis(3)`

svr_enable_vpp

Enables preprocessing of Verilog files.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables or disables the preprocessing of Verilog files.

By default, this variable is set to **false**, which disables the preprocessing of Verilog files.

If you set this variable to **true** before you run the **read_verilog** command, the Verilog preprocessor detects and expands the following Verilog directives:

- `define
- `else
- `endif
- `ifdef
- `include
- `undef

The preprocessor creates intermediate files in the directory that is specified by the **pt_tmp_dir** variable. Also, the `include directive uses the **search_path** variable to find files.

Very few structural Verilog files use preprocessor directives. Set this variable to **true** only if your Verilog file contains these directives.

SEE ALSO

```
read_verilog(2)
pt_tmp_dir(3)
search_path(3)
```

svr_keep_unconnected_nets

Specifies whether to preserve or discard unconnected nets.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable is used only by the native Verilog reader. You can set this variable to one of these values:

- **true** (the default) - Preservies unconnected nets.
- **false** - Discards unconnected nets.

SEE ALSO

`read_verilog(2)`

synopsys_program_name

Indicates the name of the program currently running.

TYPE

string

DESCRIPTION

This variable is read only, and is set by the application to indicate the name of the program you are running. This is useful when writing scripts that are mostly common between some applications, but contain some differences based on the application.

To determine the current value of this variable, use **get_app_var synopsys_program_name**.

SEE ALSO

get_app_var(2)

synopsys_root

Indicates the root directory from which the application was run.

TYPE

string

DESCRIPTION

This variable is read only, and is set by the application to indicate the root directory from which the application was run.

To determine the current value of this variable, use **get_app_var synopsys_root**.

SEE ALSO

get_app_var(2)

tcl_interactive

Indicates whether the Tcl interpreter is operating in interactive mode.

TYPE

string

DEFAULT

0

DESCRIPTION

This global variable indicates whether the Tcl interpreter is operating in interactive mode. If the variable is set to 1 , then the interpreter operates as an interactive interpreter.

You cannot change the setting of this read-only Tcl variable.

tcl_library

Specifies the path to the Tcl library.

TYPE

string

DEFAULT

The default path is determined by your tool installation.

DESCRIPTION

You cannot change the setting of this read-only Tcl variable.

tcl_pkgPath

Specifies the path to the Tcl library.

TYPE

string

DEFAULT

The default path is determined by your tool installation.

DESCRIPTION

You cannot change the setting of this read-only Tcl variable.

timing_all_clocks_propagated

Determines whether or not all clocks are created as propagated clocks.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), clocks are created as nonpropagated clocks. If you set this variable to **true**, all clocks subsequently created by the **create_clock** or **create_generated_clock** command are created as propagated clocks.

By default, the **create_clock** and **create_generated_clock** commands 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, the **create_clock** and **create_generated_clock** commands 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 nonpropagated).

SEE ALSO

`create_clock(2)`

```
create_generated_clock(2)
set_propagated_clock(2)
```

timing_allow_short_path_borrowing

Enables time borrowing through level sensitive latches for hold time checks.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable affects time borrowing for short paths (used for hold checks) at a level-sensitive latch. By default, PrimeTime 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.

To enable borrowing for short paths, set the **timing_allow_short_path_borrowing** variable to **true**.

SEE ALSO

```
report_timing(2)
set_max_time_borrow(2)
```

timing_aocvm_analysis_mode

Configures the depth calculation of an advanced on-chip variation (AOCV) analysis.

TYPE

string

DEFAULT

separate_launch_capture_depth

DESCRIPTION

The **timing_aocvm_analysis_mode** variable specifies one of three methods to calculate path metrics during AOCV analysis. The path metrics consist of depth and distance values:

- Depth is used to index the random component of variation in an AOCV derating table. Depth is defined as the number of cell (or net) delay timing arcs in a path from the path common point. Random coefficients affect the depth calculation. For more information, see the **set_aocvm_coefficient** man page.
- Distance is used to index the systematic component of variation in an AOCV derating 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. 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.

To specify the analysis mode, set the **timing_aocvm_analysis_mode** variable to one of these values:

- **separate_launch_capture_depth** - Calculates separate depth values for launch and capture paths. The launch clock and data paths are considered together, and the capture clock path is considered separately.
- **combined_launch_capture_depth** - Considers launch and capture depths together and calculates

a combined depth for the entire path.

- **separate_data_and_clock_metrics** - Calculates separate depths for the clock and data paths and uses the appropriate AOCV delay deratings based on the separate depths.

The AOCV path metric calculation is further influenced by the **timing_aocvm_enable_clock_network_only**, **timing_aocvm_enable_single_path_metrics**, and **timing_ocvm_enable_distance_analysis** variables.

SEE ALSO

```
get_timing_paths(2)
read_aocvm(2)
remove_aocvm(2)
report_aocvm(2)
report_timing(2)
set_aocvm_coefficient(2)
timing_ocvm_enable_distance_analysis(3)
timing_aocvm_enable_clock_network_only(3)
timing_aocvm_enable_single_path_metrics(3)
```

timing_aocvm_enable_analysis

Enables graph-based advanced on-chip variation (AOCV) analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **false** (the default), the tool does not perform graph-based advanced on-chip variation (AOCV) timing updates. A path-based advanced OCV analysis can be performed in this mode using the **-pba_mode** option of 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. You should specify constant derates that do not clip the range of the path-based advanced OCV derates to avoid optimism.

When you set this variable to **true**, the graph-based advanced OCV timing update is performed as part of the **update_timing** command. A path-based advanced OCV 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 advanced OCV derates computed during the **update_timing** command tightly bound the path-based advanced OCV derates without clipping their range. Note that setting this variable to **true** automatically switch the design into **on_chip_variation** analysis mode using the **set_operating_conditions** command.

SEE ALSO

```
get_timing_paths(2)
read_aocvm(2)
report_aocvm(2)
report_timing(2)
set_operating_conditions(2)
```

timing_aocvm_enable_clock_network_only

Configures the advanced on-chip variation (AOCV) analysis to be applied in the clock network only.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable specifies whether AOCV analysis is performed on all timing arcs of the chip or only for timing arcs in the clock network.

- **true** - Applies AOCV derating to arc delays in the clock network only; calculates clock network AOCV depth and distance metrics 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.
- **false** (the default) - Applies AOCV derating to all arc delays and calculates AOCV depth and distance metrics by considering both clock and data network topology.

SEE ALSO

```
timing_aocvm_analysis_mode(3)
timing_aocvm_enable_single_path_metrics(3)
timing_ocvm_enable_distance_analysis(3)
```

timing_aocvm_enable_single_path_metrics

Prevents the use of separate depth and distance values for nets and cells during advanced on-chip variation (AOCV) analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable determines whether AOCV analysis considers depth and distance metrics for cell arcs and net arcs separately.

- **true** - Calculates and uses only cell arc metrics to look up both cell and net deratings. This behavior is backward compatible with the legacy Tcl-based "LOCV" solution.
- **false** (the default) - Uses separate depth and distance values for nets and cells during AOCV analysis. The tool uses cell metrics when looking up cell deratings and net metrics when looking up net deratings.

SEE ALSO

```
timing_aocvm_analysis_mode(3)
timing_aocvm_enable_clock_network_only(3)
timing_ocvm_enable_distance_analysis(3)
```

timing_aocvm_ocv_precedence_compatibility

Controls the fallback to on-chip variation (OCV) deratings when advanced on-chip variation (AOCV) is enabled.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable is obsolete and provided for backward compatibility only. Use the **timing_ocvm_precedence_compatibility** variable instead.

SEE ALSO

`timing_ocvm_precedence_compatibility(3)`

timing_arc_attributes

Describes the predefined application attributes for timing_arc objects.

DESCRIPTION

annotated_delay_delta_max

Type: float
Returns the maximum of the annotated_delay_delta_max_rise and annotated_delay_delta_max_fall attributes.

annotated_delay_delta_max_fall

Type: float
Returns the delay that is added to the maximum falling delay of the timing arc. The additional delay is set by either the set_annotated_delay -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_delay_delta_max_rise

Type: float
Returns the delay that is added to the maximum rising delay of the timing arc. The additional delay is set by either the set_annotated_delay -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_delay_delta_min

Type: float
Returns the minimum of the annotated_delay_delta_min_rise and annotated_delay_delta_min_fall attributes.

annotated_delay_delta_min_fall

Type: float
Returns the delay that is added to the minimum falling delay of the timing arc. The additional delay is set by either the set_annotated_delay -delta_only command or by PrimeTime SI during crosstalk analysis.

annotated_delay_delta_min_rise

Type: float
Returns the delay that is added to the minimum rising delay of the timing arc. The additional delay is set

by either the set_annotated_delay -delta_only command or by PrimeTime SI during crosstalk analysis.

delay_max

Type: float
Returns the maximum of the delay_max_rise and delay_max_fall attributes.

delay_max_fall

Type: float
Returns the maximum falling delay of the timing arc.

delay_max_rise

Type: float
Returns the maximum rising delay of the timing arc.

delay_min

Type: float
Returns the minimum of the delay_min_rise and delay_min_fall attributes.

delay_min_fall

Type: float
Returns the minimum falling delay of the timing arc.

delay_min_rise

Type: float
Returns the minimum rising delay of the timing arc.

from_pin

Type: collection
Returns a collection containing the from pin of the timing arc.

is_annotated_fall_max

Type: boolean
Returns true if the maximum falling delay of the timing arc is back-annotated.

is_annotated_fall_min

Type: boolean
Returns true if the minimum falling delay of the timing arc is back-annotated.

is_annotated_rise_max

Type: boolean
Returns true if the maximum rising delay of the timing arc is back-annotated.

is_annotated_rise_min

Type: boolean
Returns true if the minimum rising delay of the timing arc is back-annotated.

is_cellarc

Type: boolean
Returns true if the timing arc is a cell arc, and false for net arcs.

is_constraint_scaled_max_fall

Type: boolean
Returns true if the constraint timing arc uses scaling between libraries for the driver model for maximum fall analysis.

is_constraint_scaled_max_rise

Type: boolean
Returns true if the constraint timing arc uses scaling between libraries for the driver model for maximum rise analysis.

is_constraint_scaled_min_fall

Type: boolean
Returns true if the constraint timing arc uses scaling between libraries for the driver model for minimum fall analysis.

is_constraint_scaled_min_rise

Type: boolean
Returns true if the constraint timing arc uses scaling between libraries for the driver model for minimum rise analysis.

is_db_inherited_disabled

Type: boolean
Returns true if the arc is a .db database-inherited disabled arc. Such an arc has been disabled for loop breaking by upstream tools. To be consistent, the PrimeTime tool disables these same arcs if the timing_keep_loop_breaking_disabled_arcs variable is set to true.

is_disabled

Type: boolean
Returns true if the timing arc is disabled.

is_user_disabled

Type: boolean
Returns true if the timing arc is disabled by using the set_disable_timing command.

mode

Type: string
Returns the mode string of the timing arc.

object_class

Type: string
Returns the class of the object, which is the string "timing_arc". You cannot set this attribute.

sdf_cond

Type: string
Returns the SDF condition of the timing arc.

sdf_cond_end

Type: string
Returns the SDF condition at the endpoint of the timing arc. Variable sdf_enable_cond_start_end must be true.

sdf_cond_start

Type: string
Returns the SDF condition at the startpoint of the timing arc. Variable sdf_enable_cond_start_end must be true.

sense

Type: string
Returns the sense of the timing arc.

to_pin

Type: collection
Returns a collection containing the "to" pin of the timing arc.

when

Type: string
Returns the "when" string of the timing arc.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

timing_bidirectional_pin_max_transition_check

Specifies the extent of max transition design rule checks on bidirectional pins.

TYPE

string

DEFAULT

both

DESCRIPTION

This variable specifies the extent of a max transition design rule checks for bidirectional pins. Set the variable to one of these values:

- **both** (the default) - Checks the driver and load.
- **driver** - Checks the driver.
- **load** - Checks the load.

timing_calculation_across_broken_hierarchy_cc

Specifies whether delay calculation ignores hierarchical pins where clock constraints are specified when performing detailed RC calculation.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to one of the following values:

- **false** (the default) - Computes the interconnect delay from leaf pin to leaf pin; ignores intervening hierarchical pins where clock constraints are defined.
- **true** - Attempts to use the hierarchical pin during calculation.

SEE ALSO

```
create_clock(2)
create_generated_clock(2)
report_delay_calculation(2)
```

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 partial_input_delay unconstrained_endpoints unexpandable_clocks no_driving_cell pulse_clock_non_pulse_clock_merge pll_configuration

DESCRIPTION

Defines the default checks to be performed when you run the **check_timing** command without any options or with the **-include** or **-exclude** option.

You can change this variable setting before you run the **check_timing** command. Alternatively, you can override the default checks by running the **check_timing** command with the **-override_defaults** option.

SEE ALSO

`check_timing(2)`

timing_clock_gating_check_fanout_compatibilit

Controls whether the effects of the **set_clock_gating_check** command propagates through logic, or applies only to the specified design object.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls the behavior of the **set_clock_gating_check** command when it is applied to design netlist objects (ports, pins or cells).

If this variable is set to **false**, PrimeTime uses the current behavior where the effects of the **set_clock_gating_check** command apply only to the specified design objects, and do not propagate through the transitive fanout. When this behavior is enabled, specifying the command on a port has no effect. This behavior is consistent with Design Compiler and IC Compiler.

To apply the clock gating settings to an entire clock domain without enumerating all gating cells, clock objects can be provided to the **set_clock_gating_check** command. When clock objects are provided, the clock gating settings apply to all instances of clock gating for those clocks.

If this variable is set to **true**, PrimeTime uses the behavior from older versions where the **set_clock_gating_check command** also applies to the transitive fanout of the specified design objects. There is no way to configure only the specified design objects without also propagating the clock gating settings to downstream logic.

SEE ALSO

set_clock_gating_check(2)

timing_clock_gating_propagate_enable

Allows the gating enable signal delay to propagate through the gating cell.

TYPE

integer

DEFAULT

true

DESCRIPTION

You can set this variable to one of these values:

- **true** (the default) - Allows the delay and slew from the data path of the gating check to propagate. If the output goes to a data pin, setting this variable to **true** produces the most desirable behavior.
- **false** - Prevents the delay and slew from the data path of the gating check from propagating. Only the delay and slew from the clock path is propagated. If the output goes to a clock pin of a latch, setting this variable to **false** produces the most desirable behavior.

timing_clock_reconvergence_pessimism

Selects 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.

Set this variable to one of the following values:

- **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.
- **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. 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**,the CRPR for min pulse width checks will be computed at the last common pin where the launch and capture edges match. The result will be zero if a common pin can not be found.

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
timing_remove_clock_reconvergence_pessimism(3)
```

timing_crpr_different_transition_derate

Specifies the fraction of the common clock reconvergence pessimism (CRP) that is removed if the transitions on the launching and capturing clock paths are different.

TYPE

float

DEFAULT

1.0

DESCRIPTION

When computing the common clock pessimism, the path to the common point is assumed to be correlated.

If the "normal" setting of the variable **timing_clock_recovergence_pessimism** is used, a pessimism value is computed for both the rising path and falling path to the common point, and the minimum is removed. This assumes the launch and capture paths, despite different transitions, are fully correlated.

However, the paths are actually only partially correlated. The gates and wires are the same, but the paths through the transistors of the gates may be different.

To reduce the amount of pessimism removed, set this variable to a factor less than 1.0. For example, setting it to 0.90 causes 90 percent of the calculated pessimism to be removed, resulting less reported slack and a more conservative analysis than removing 100 percent of the calculated pessimism.

Note that setting a factor of 0.0 results in no pessimism being removed for different transitions. This is not equivalent to using the "same_transition" setting for the **timing_clock_recovergence_pessimism** variable. In that setting, the tool continues to search for a common point with matching transitions. With the "normal" setting, the topological common point is always selected, irrespective of the whether the transitions match.

In parametric on-chip variation analysis, this variable derates just the nominal value of the common clock pessimism. To derate the variation, use the variable **timing_crpr_different_transition_variation_derate**.

SEE ALSO

```
timing_remove_clock_reconvergence_pessimism(3)
timing_clock_reconvergence_pessimism(3)
timing_crpr_different_transition_variation_derate(3)
```

timing_crpr_different_transition_variation_dera

Specifies the fraction of the variation of the common clock reconvergence pessimism (CRP) that is removed if the transitions on the launching and capturing clock paths are different.

TYPE

float

DEFAULT

0.0

DESCRIPTION

Parametric on-chip variation (POCV) analysis internally computes arrival, required, and slack values based on statistical distributions. The common clock pessimism removes both nominal and variation pessimism from the slack.

When computing the common clock pessimism, if the launch and capture transition (rise/fall) at the common point are the same, the variation at the common point is removed from the slack.

If the "normal" setting of the variable **timing_clock_recovergence_pessimism** is used, and the launch and capture transition (rise/fall) at the common point are different, the variation will not be removed. This is because the rising path and the falling path are not fully correlated.

If this is too pessimistic, a percentage of the variation can be removed by setting the variable **timing_crpr_different_transition_variation_derate** to a non-zero value. i.e. to remove 10% of the variation, use the value 0.1. To allow the full credit of the variation, use 1.0.

SEE ALSO

```
timing_pocvm_enable_analysis(3)
timing_remove_clock_reconvergence_pessimism(3)
timing_clock_reconvergence_pessimism(3)
timing_crpr_different_transition_derate(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

If this variable is set to **true**, CRP is removed for all paths that fan out directly from clock source pins to the data pins of sequential devices.

If 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 is 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
```

Note:
When this variable is set to **true**, all sequential devices that reside in the fanout of clock source pins must be handled separately in the subsequent timing update. This might cause severe performance degradation to the timing update.

SEE ALSO

timing_remove_clock_reconvergence_pessimism(3)

timing_crpr_remove_muxed_clock_crp

Controls whether clock reconvergence pessimism removal (CRPR) considers common path reconvergence between related clocks.

TYPE

Boolean

DEFAULT

true

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**, the separate clock paths up to the multiplexer are treated as reconvergent, and the CRP includes the reconvergence point as well as any downstream common logic. If this variable is set to **false**, the common pin is 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 is **true**, which removes the additional 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

5

DESCRIPTION

This variable specifies the amount of pessimism that clock reconvergence pessimism removal (CRPR) is allowed to leave in the report. The unit is in picoseconds (ps), regardless of the units of the main library. The minimum allowed value is 1 picosecond.

The threshold is per reported slack; setting 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 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.

In most cases, the default setting provides a reasonable tradeoff between accuracy and runtime. You can use different settings throughout the design cycle: larger during the design phase, smaller for signoff. You might need to experiment and set a different value when moving to a different technology.

SEE ALSO

timing_remove_clock_reconvergence_pessimism(3)

timing_disable_bus_contention_check

Disables checking for timing violations resulting from transient contention on design buses.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable applies only to bus designs that have multiple three-state drivers.

When set to **false** (the default), PrimeTime reports these timing violations. When set to **true**, PrimeTime ignores timing setup and hold (max and min) violations that occur as a result of transient bus contention.

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 never occur. By setting the value to **true**, you guarantee that on a multidriven 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 **timing_disable_bus_contention_check** variable is **false**. The **timing_disable_bus_contention_check** and **timing_disable_floating_bus_check** variables 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 three_state_disable and three_state_enable timing arc types. These timing arcs connect the enable pin to

the output pin of the three-state buffers. For more information, see the Library Compiler documentation.

SEE ALSO

`timing_disable_floating_bus_check(3)`

timing_disable_clock_gating_checks

Disables checking for setup and hold clock gating violations.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), PrimeTime automatically determines clock-gating and performs clock-gating setup and hold checks.

If you set this variable to **true**, PrimeTime disables clock-gating setup and hold checks.

SEE ALSO

`report_constraint(2)`
`set_clock_gating_check(2)`

timing_disable_cond_default_arcs

Disables the default, nonconditional timing arc between pins that have conditional arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, this variable disables nonconditional timing arcs between any pair of pins that have at least one conditional arc. When this variable is set to **false**, 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.

If the specified conditions cover all possible state-dependent delays, set this variable to **true**, so that the default arc is not used. For example, consider a 2-input XOR gate with inputs A and B and output Z. If the delays between A and Z are specified with two arcs with respective conditions 'B' and 'B~", the default arc between A and Z is not used and should be disabled.

SEE ALSO

`report_disable_timing(2)`

timing_disable_floating_bus_check

Disables checking for timing violations resulting from transient floating design buses.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable applies only to bus designs that have multiple three-state drivers. When set to **true**, PrimeTime ignores timing setup and hold (max and min) violations that occur as a result of transient floating buses. When set to **false** (the default), PrimeTime reports these timing violations.

The 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 never occur. By setting the value to **true**, you guarantee that on a multidriven 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 **timing_disable_bus_contention_check** variable is **false**. The **timing_disable_floating_bus_check** and **timing_disable_bus_contention_check** variables 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 three_state_disable and three_state_enable timing arc types. These timing arcs connect the enable pin to the output pin of the three-state buffers. For more information, see the Library Compiler documentation.

SEE ALSO

timing_disable_bus_contention_check(3)

timing_disable_internal_inout_cell_paths

Enables bidirectional feedback paths within a cell.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If this variable is set to **true** (the default), PrimeTime automatically disables bidirectional feedback paths in a cell. If you set this variable to **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 **timing_disable_internal_inout_net_arcs** variable.

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

If this variable is set to **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.

If you set this variable to **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 **timing_disable_internal_inout_cell_paths** variable.

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

Disables the timing analysis of recovery and removal checks in the design.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), PrimeTime performs recovery and removal checks. For a description of these checks, see the man page for the **report_constraint** command.

To disable recovery and removal timing analysis, set this variable to **true**.

SEE ALSO

`report_constraint(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

The following description assumes that the data paths of interest are setup paths, because it refers specifically to time borrowing scenarios. However, if the **timing_allow_short_path_borrowing** variable is **true**, the same description 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 that originate at the D-pin are created.

Sometimes there is a difference between the launching and capturing latch latencies, due to either reconvergent paths in the clock network or different minimum and maximum 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 therefore does not even "see" the clock edge at the latch.

When the **timing_early_launch_at_borrowing_latches** variable is **true** (the default), the tools eliminates this pessimism by using the early latch latency to launch such paths. Note that only paths that originate from a latch D-pin are affected. When the variable is set to **false**, the tools uses late clock latency to launch all setup paths in the design.

When the **timing_early_launch_at_borrowing_latches** and **timing_remove_clock_reconvergence_pessimism** variables are both set to **true**, the tool applies CRPR to paths that do not start from transparent latch D-pins, while the paths that start from transparent latch D-pins have early launch using the minimum path. It is not possible to apply both pessimism removal techniques on the same timing path.

The following recommended settings depend on the characteristics of your design:

- On designs with long clock paths, consider setting **timing_early_launch_at_borrowing_latches** to **false**. This allows CRPR to be applied on paths that start from transparent latch D-pins. When clock paths are long, CRPR can be a more powerful pessimism reduction technique.
- On designs with shorter common clock paths, or where critical paths traverse several latches, consider setting **timing_early_launch_at_borrowing_latches** to **true**. This results in more pessimism removal, even though paths that start from transparent D-pins do not get any CRPR credit.

SEE ALSO

```
report_timing(2)
timing_allow_short_path_borrowing(3)
timing_remove_clock_reconvergence_pessimism(3)
```

timing_enable_auto_mux_clock_exclusivity

Enables automatic inference of MUX cells for clock exclusivity.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, the tool automatically identifies the MUX cells on the clock network and forces the clocks that go through different data lines of the MUX cells to be exclusive.

By default, the variable is set to **false** and the tool does not automatically infer the exclusivity of clocks from MUX cells in the clock network. Please note that if there is a generated clock defined after the exclusivity point, the exclusivity states will be lost. This is similar to the existing behavior of the generated clocks. When a generated clock is created at a pin, all other clocks arriving at that pin are blocked unless they also have generated clock versions created at that pin.

Please note that this feature is not currently supported in **extract_model**, **characterize_context** and **write_eco_design** commands.

SEE ALSO

```
set_clock_exclusivity(2)
set_disable_auto_mux_clock_exclusivity(2)
```

```
report_clock(2)
```

timing_enable_clock_propagation_through_pres

Enables propagation of clock signals through preset and clear pins.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, clock signals propagate through the preset and clear pins of a sequential device. This occurs only if clock signals are incident on such pins.

If CRPR is enabled, the tool considers any sequential devices in the fanout of such pins for analysis.

SEE ALSO

`timing_remove_clock_reconvergence_pessimism(2)`

timing_enable_clock_propagation_through_thre

Allows the clocks to propagate through the enable pin of a three-state cell.

TYPE

int

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), PrimeTime does not propagate clocks between a pair of pins if there is at least one timing arc with a disable sense between those pins.

To allow the clocks to propagate through the enable pins of three-state cells, set this variable to **true**.

timing_enable_constraint_variation

Enables constraint variation in parametric on-chip variation (POCV) analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set this variable to **true** and set the **timing_pocvm_enable_analysis** variable to **true**, constraint variation is considered in parametric on-chip variation (POCV) analysis.

SEE ALSO

```
get_timing_paths(2)
read_aocvm(2)
report_aocvm(2)
report_timing(2)
set_operating_conditions(2)
timing_pocvm_enable_analysis(3)
```

timing_enable_cross_voltage_domain_analysis

Enables reduced-pessimism analysis of timing paths that cross multiple voltage domains.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to true to enable reduced-pessimism analysis of the timing paths that cross multiple voltage domains. Standard on-chip variation analysis leads to pessimistic results for cross-domain paths because it can consider the behavior of different cells operating at both the high voltage and low voltage within a given domain at the same time. After single-domain path violations are found and fixed using standard on-chip variation analysis, you can analyze just the cross-domain paths and reduce the pessimism of the results by setting this variable set to true.

When this variable is set to true, the **update_timing** command identifies paths that traverse multiple voltage domains and subsequently limits the reporting to only those paths. If a path-based analysis is performed by using the **-pba_mode path** option of the **report_timing** command, PrimeTime performs an efficient path-based recalculation of the cross-domain paths. This analysis finds the worst-case voltage for each domain crossed by each path, but eliminates the pessimism that occurs in standard on-chip variation analysis.

If you do not have any reliable characterization information with respect to voltage variation, you can use derating to analyze the impact of different supply voltages. The **set_cross_voltage_domain_analysis_guardband** command sets derating factors that apply whenever the **timing_enable_cross_voltage_domain_analysis** variable is set to true.

SEE ALSO

```
report_timing(2)
set_cross_voltage_domain_analysis_guardband(2)
update_timing(2)
```

timing_enable_cumulative_incremental_derate

Enables or disables accumulating the incremental derates.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, if the -increment option is used with the set_timing_derate command, PrimeTime will use only the last increment set. If you set this variable to **true**, PrimeTime will accumulate all the -increment set by the user.

SEE ALSO

`set_timing_derate(2)`

timing_enable_library_max_cap_lookup_table

Specifies whether the frequency-indexed lookup table values for max_capacitance, if they exist, are to be used.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If this variable is set to **true** (the default), and a max_capacitance lookup table is defined on the library pin, the lookup table is used to determine the max_capacitance value applied at the pin. The lookup table is indexed by clock frequency, the inverse of the clock period. The actual max_capacitance applied is the most restrictive max_capacitance constraint for the specified pin. Specifically, the smallest max_capacitance value is taken among:

- The lookup table-derived value
- The library-cell max_capacitance value
- All user-specified max_capacitance values (at the pin, port, clock, or design level) at the pin

To ignore the lookup table-derived value, set this variable to **false**.

SEE ALSO

```
report_constraint(2)
set_max_capacitance(2)
timing_enable_max_cap_precedence(3)
timing_library_max_cap_from_lookup_table(3)
```

timing_enable_max_cap_precedence

Enables precedence rules for max_capacitance constraints.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If the **timing_enable_max_cap_precedence** variable is set to **false** (the default), the most restrictive (smallest) max_capacitance pertaining to a pin is applied.

If you set the **timing_enable_max_cap_precedence** variable to **true**, the pin-level max_capacitance value takes precedence over the library-derived max_capacitance value, which in turn take precedence over design-level max capacitance value.

SEE ALSO

```
report_constraint(2)
set_max_capacitance(2)
timing_library_max_cap_from_lookup_table(3)
```

timing_enable_max_capacitance_set_case_ana

Enables the checking of the max capacitance constraint on constant pins.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

To check the max capacitance constraint for constant pins, set this variable to **true**.

SEE ALSO

report_constraint(2)
timing_enable_max_transition_set_case_analysis(3)

timing_enable_max_transition_set_case_analys

Enables the checking of the max transition constraint on set case constant pins.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

To check the max transition constraint for set case constant pins, set this variable to **true**.

SEE ALSO

report_constraint(2)
timing_enable_max_capacitance_set_case_analysis(3)

timing_enable_normalized_slack

Enables or disables normalized slack analysis during timing updates.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to *true*, the tool performs normalized slack analysis during timing updates. If set to **false** (the default), the tool does not perform normalized slack analysis and reporting.

Normalized slack analysis is an additional optional analysis that computes a normalized slack for each timing path. The normalized slack is the slack divided by the idealized allowed propagation delay. For paths in a single clock domain, the allowed propagation delay is the time between two different edges of the clock. For 50% duty cycle clocks, the allowed propagation delay is usually an integer number of half periods.

Normalized slack can be used to determine the paths which limit the clock frequency. Normalized slack prioritizes violating paths allowed few clock cycles. Fixing these paths first allows the most improvement in the clock period.

If normalized slack analysis is enabled during update timing, paths can be gathered and reported using normalized slack, using the **report_timing -normalized_slack** and **get_timing_paths -normalized_slack** commands.

SEE ALSO

report_timing(2)

timing_enable_preset_clear_arcs

Enables preset and clear arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **false** (the default), PrimeTime disables all preset and clear timing arcs.

To enable asynchronous preset and clear timing arcs for use during timing path analysis, set this variable to **true**.

The tool performs minimum pulse width checks defined on asynchronous preset and clear pins, regardless of this variable setting.

SEE ALSO

`report_timing(2)`

timing_enable_pulse_clock_constraints

Enables the checking of pulse clock constraints.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable determines if pulse clock constraints are checked or not.

When this variable is **true**, the tool checks the pulse clock constraints set by the **set_pulse_clock_min_width**, **set_pulse_clock_max_width**, **set_pulse_clock_min_transition**, and **set_pulse_clock_max_transition** commands.

When this variable is **true**, the min pulse width constraints set by the **set_min_pulse_width** command do not apply to pulse clock networks and more specific pulse clock constraints checked.

SEE ALSO

```
report_constraint(2)
report_pulse_clock_max_transition(2)
report_pulse_clock_max_width(2)
report_pulse_clock_min_transition(2)
report_pulse_clock_min_width(2)
set_pulse_clock_max_transition(2)
```

```
set_pulse_clock_max_width(2)
set_pulse_clock_min_transition(2)
set_pulse_clock_min_width(2)
```

timing_enable_slew_variation

Enables transition variation in parametric on-chip variation (POCV) analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set this variable to **true** and set the **timing_pocvm_enable_analysis** variable to **true**, transition variation is considered in parametric on-chip variation (POCV) analysis.

SEE ALSO

```
get_timing_paths(2)
read_aocvm(2)
report_aocvm(2)
report_timing(2)
set_operating_conditions(2)
timing_pocvm_enable_analysis(3)
```

timing_enable_through_paths

Enables or disables advanced analysis and reporting through transparent latches.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, this variable enables advanced analysis through transparent latches during timing updates and reporting. When set to **false** (the default), the tool disables advanced analysis and reporting through transparent latches.

By default, the tool analyzes and reports paths through transparent latches as a series of path segments between latches. These segments can be reported together using the **report_timing** option **-trace_latch_borrow**. Max pin slacks (**max_rise_slack**, **max_fall_slack**) for a pin in the design can be affected by borrowing latches in the fanin of the pin, but are not affected by timing calculations in parts of the design past the first level of latches in the fanout of the pin.

If you enable advanced analysis through transparent latches, you can report paths through latches as a single timing path. Pin slacks can be affected by timing calculations past the first level of latches in the fanout. In addition, you can report specific paths through latches by using the **-from**, **-through**, and **-to** options of **report_timing**, where the options specify objects that are separated by one or more transparent latches.

The advanced analysis is limited when there are latch loops in the design. The tool chooses specific latch data pins in the loops to act as loop breaker latches. For these latch data pins, the behavior is the same as if the **timing_enable_through_paths** variable was set to **false**. Reporting through these special latch data pins is not supported. The tool automatically selects which latch data pins to act as loop breaker latches. You can guide the selection using the **set_latch_loop_breaker** command. Because of the

runtime associated with the advanced analysis, by default the tool also selects some latch data pins outside loops to have the same behavior as if **timing_enable_through_paths** was **false**. You can use the **timing_through_path_max_segments** variable to control the selection of these pins.

SEE ALSO

```
report_timing(2)
set_latch_loop_breaker(2)
timing_through_path_max_segments(3)
```

timing_gclock_source_network_num_master_r

Specifies the maximum number of register clock pins clocked by the master clock allowed in generated clock source latency paths.

TYPE

integer

DEFAULT

10000000

DESCRIPTION

This variable specifies 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 the primary master clock differs from the generated clock when source latency paths are being computed.

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

This variable specifies a transition value to use at clock pins of a flip-flop. If you run the **set_clock_transition** command, and the clock is ideal, the transition value is used, and this variable has no effect. If the clock transition is not set by the **set_clock_transition** command, and the clock is ideal, then this variable has the following effect:

- **true** (the default) - Uses a zero transition value for ideal clocks.
- **false** - Uses a propagated transition value.

If the clock is ideal, the **set_clock_transition** command overrides the effect of this variable.

SEE ALSO

report_delay_calculation(2)
set_clock_transition(2)

timing_include_available_borrow_in_slack

Specifies whether PrimeTime includes available borrow time in slack.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **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.

If you set this variable to **true**, any path terminating at the data pin of a transparent latch has positive or negative slack measured with respect to the closing transition at the latch. That is, available borrow time is considered 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.

If this variable is set to **true**, slack results can be misleadingly optimistic in the sense that a pin can appear to have positive slack, but increasing delay at the pin can lead to additional borrowing and timing violations downstream of the latch. This is a serious problem for ECO flows. Do not enable this variable when running ECO flows, or PrimeTime ECO in particular. If the variable is enabled during PrimeTime ECO fixing, additional setup violations can occur.

For most situations, rather than using **timing_include_available_borrow_in_slack**, it is better to use advanced latch analysis, by enabling **timing_enable_through_paths**. The advanced latch analysis provides slack calculations through most latches, without optimism for ECO flows.

SEE ALSO

```
report_timing(2)
set_max_time_borrow(2)
```

timing_include_uncertainty_for_pulse_checks

Specifies how the clock uncertainty is applied to minimum period and minimum pulse width checks.

TYPE

string

DEFAULT

setup_hold

DESCRIPTION

To specify how the clock uncertainty is applied for minimum period and minimum pulse width checks, set this variable to one of the following values:

- **setup_hold** - Applies the worst setup or hold uncertainty.
- **setup_only** - Applies only the setup uncertainty.
- **hold_only** - Applies only the hold uncertainty.
- **none** - Applies neither the setup nor hold uncertainty.

Use the **set_clock_uncertainty** command to specify the uncertainty for a specified clock. If you use the command with neither the **-setup** nor **-hold** option, then the clock uncertainty is applied to both setup and hold checks.

SEE ALSO

```
report_constraint(2)
report_min_period(2)
report_min_pulse_width(2)
set_clock_uncertainty(2)
```

timing_input_port_default_clock

Specifies whether a default clock is assumed at input ports for which you have not defined a clock with the **set_input_delay** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable affects the behavior of PrimeTime when you set an input delay without a clock on an input port.

If you set this variable to **true**, 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. Also, the period of this clock is equal to the base period of all these related clocks. PrimeTime SI excludes victim and aggressor arrival windows associated with this imaginary clock for crosstalk analysis.

If this variable is set to **false**, no such imaginary clock is assumed.

SEE ALSO

`set_input_delay(2)`

timing_is_clock_pin_attribute_on_clock_source

When set to **true**, the value of pin attribute "is_clock_pin" will be **true** for clock source pins/ports, otherwise, the value of the attribute will be **false**

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By definition, for a design instance pin object, the **"is_clock_pin"** attribute is **true** if the pin is a valid and active clock pin in the design; that is, it is reached by a clock signal, and the sequential cell instance containing this pin is not disabled by disabled timing arcs or by case analysis. When clock source is defined on cell instance pin that is not clock pin, such as the Q output, PrimeTime no longer marks the pin with **"is_clock_pin"** attribute, and is now consistent with IC Compiler II.

Please note that the variable only affects the value of the attribute **"is_clock_pin"**, and therefore, can be set after update_timing.

SEE ALSO

`get_attribute(2)`
`report_attribute(2)`

timing_keep_loop_breaking_disabled_arcs

Specifies 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 inherited 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 can be removed individually without affecting the other .db inherited 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 linking, it has no effect.

To remove .db inherited arcs after they are accepted, use the **remove_disable_timing** command because they are user-defined.

The **is_db_inherited_disabled** timing_arc attribute indicates whether an arc is a .db inherited disabled arc.

To remove all .db inherited disable timing arcs for loop breaking, use this command:

```
pt_shell> remove_disable_timing [get_timing_arcs -of [get_cell *] \
          -filter "is_db_inherited_disabled == true"]
```

SEE ALSO

remove_disable_timing(2)
report_disable_timing(2)

timing_keep_waveform_on_points

Keeps the waveform data from advanced waveform propagation on timing points when timing paths are created by the **get_timing_paths** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the tool does not keep the waveform data from advanced waveform propagation on timing points because most timing analysis flows do not use this data. This default behavior reduces the peak memory usage.

To override the default behavior and keep the waveform data on timing points during the **get_timing_paths** command, set the **timing_keep_waveform_on_points** variable to **true**. Use this variable setting when

- You use the **write_spice_deck** command. This keeps the waveform data on timing points so that the corresponding waveform can be written to the SPICE deck.
- You want to see the exact waveform in the PrimeTime GUI.

Note: This variable does not affect the advanced waveform propagation mode. This variable is only for timing paths created by the **get_timing_paths** command after a full timing update. After you change this variable setting, a full timing update is not required.

SEE ALSO

```
get_timing_paths(2)
write_spice_deck(2)
```

timing_lib_cell_derived_clock_name_compatibi

Specifies how generated clock names are derived from library files.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable specifies how generated clock names are derived from library files; you must set this variable before linking the design:

- **true** - Derives the name of the generated clock from the name of its source or the first source name in case of multisource generated clocks.
- **false** - Takes the generated clock name directly taken from the explicit specification in the library.

SEE ALSO

```
link_design(2)
report_clock(2)
```

timing_library_max_cap_from_lookup_table

Specifies whether the frequency-indexed lookup table values for max_capacitance, if they exist, override other library-derived max_capacitance constraints.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), the **report_constraint** command uses the most restrictive max_capacitance constraint for the specified pin. Specifically, the smallest max_capacitance value is taken among:

- The lookup table-derived value
- The library-cell max_capacitance value
- All user-specified max_capacitance values (at the pin, port, clock, or design level) at the pin

If you set this variable to **true**, and a max_capacitance lookup table is defined on the library pin, and if a valid data signal reaches the pin (that is, if there is at least one clock launching a signal that arrives at the pin), all other library-cell max_capacitance values are ignored at the pin. This variable gives precedence to the frequency-based max_capacitance constraint even when this constraint is more lenient than other library-derived constraints. The tool applies the user-specified max_capacitance constraints as normal.

SEE ALSO

```
report_constraint(2)
set_max_capacitance(2)
timing_enable_library_max_cap_lookup_table(3)
timing_enable_max_cap_precedence(3)
```

timing_max_capacitance_derate

Specifies a scaling factor for design rule constraint max_capacitance violation, ranging from 1.0 to 10.0.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable relaxes the max_capacitance globally in the entire design. The max_capacitance allowed, which is the worst of library limit and user-specified values, is multiplied by this factor.

If this variable is set to 1.0 (the default), the max_capacitance violation threshold does not change.

Set **rc_ccs_extrapolation_range_compatibility** to **false** before you set this variable to a value greater than 1.0.

Setting this variable checks out a PrimeTime ADV license.

SEE ALSO

```
report_constraint(2)
rc_ccs_extrapolation_range_compatibility(3)
timing_max_transition_derate(3)
```

timing_max_normalization_cycles

Sets the upper limit for the denominator when calculating normalized slack.

TYPE

integer

DEFAULT

4

DESCRIPTION

Normalized slack analysis is an additional optional analysis that computes a normalized slack for each timing path. The normalized slack is the slack divided by the idealized allowed propagation delay. For paths in a single clock domain, the allowed propagation delay is the time between two different edges of the clock. For 50% duty cycle clocks, the allowed propagation delay is usually an integer number of half periods.

The denominator in the normalization is limited by the value of the **timing_max_normalization_cycles** variable setting multiplied by the period of the launch clock of a path. Beyond this limit, the denominator takes the value of the limit.

Limiting the denominator saves runtime during analysis. A larger value of the limit might increase runtime.

SEE ALSO

```
report_timing(2)
timing_enable_normalized_slack(3)
```

timing_max_transition_derate

Specifies a scaling factor for the design rule constraint max_transition violation, ranging from 1.0 to 1.5.

TYPE

float

DEFAULT

1.0

DESCRIPTION

This variable relaxes the max_transition globally in the entire design. The max_transition allowed, which is the worst of library limit and user-specified values on the pin, is multiplied by this factor.

If this variable is set to 1.0 (the default), the max_transition violation threshold does not change.

Set **rc_ccs_extrapolation_range_compatibility** to **false** before you set this variable to a value greater than 1.0.

Setting this variable checks out a PrimeTime ADV license.

SEE ALSO

```
report_constraint(2)
rc_ccs_extrapolation_range_compatibility(3)
timing_max_capacitance_derate(3)
```

timing_ocvm_enable_distance_analysis

Enables distance-based analysis during advanced on-chip variation (AOCV) or parametric on-chip variation (POCV) analysis.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

To control whether distance-based analysis is performed during AOCV or POCV analysis, set this variable to one of these values:

- **true** (the default) - Performs distance-based analysis if you loaded location information with the **read_parasitics** command. This information is then used to index the systematic component of variation in the derating tables.
- **false** - Does not perform distance-based analysis even if all prerequisite information is read into the current PrimeTime session.

SEE ALSO

```
read_parasitics(2)
read_parasitics_load_locations(3)
timing_aocvm_enable_analysis(3)
```

timing_pocvm_enable_analysis(3)

timing_ocvm_precedence_compatibility

Controls the fallback to on-chip variation deratings when advanced on-chip variation (AOCV) or parametric on-chip variation (POCV) is enabled.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to one of these values:

- **false** (the default) - Considers both OCV and AOCV or POCV deratings for a given object. The tool chooses the derating for graph- and path-based analysis by the following order of precedence (from highest to lowest priority):
  - OCV leaf cell derating
  - AOCV or POCV library cell derating
  - OCV library cell derating
  - AOCV or POCV hierarchical cell derating
  - OCV hierarchical cell derating
  - AOCV or POCV design derating
  - OCV design derating
- **true** - Ignores the OCV deratings for AOCV and POCV analysis.

This variable controls the deratings for both cell delays and net delays. Path-based OCV deratings set using **pba_derate_list** is not supported for AOCV or POCV analysis. The AOCV or POCV guard band derating is used only if the AOCV or POCV derating factor is used for an object.

This variable is effective only when the **timing_aocvm_enable_analysis** or **timing_pocvm_enable_analysis** variable is set to **true**.

SEE ALSO

```
set_timing_derate(2)
timing_aocvm_enable_analysis(3)
timing_pocvm_enable_analysis(3)
```

timing_path_arrival_required_attribute_include

Includes or excludes the clock edge time (the **endpoint_clock_close_edge_value** and **startpoint_clock_close_edge_value** attributes) in the **arrival** and **required** timing path attributes.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to one of the following values:

- **false** (the default) - Excludes the clock edge time of the launching or capturing clocks of the path from the calculation of the **arrival** or **required** timing path attribute, respectively.
- **true** - Includes the clock edge time of the launching or capturing clocks of the path in the calculation of the **arrival** or **required** timing path attribute, respectively. When you use this setting, the **arrival** and **required** attribute values match the arrival and required times reported by the **report_timing** command.

SEE ALSO

```
report_timing(2)
timing_path_attributes(3)
```

timing_path_attributes

Describes the predefined application attributes for timing_path objects.

DESCRIPTION

arrival

Type: float
Returns the arrival time at the endpoint of the timing path; excludes startpoint_clock_open_edge_value by default unless the timing_path_arrival_required_attribute_include_clock_edge variable is set to true.

capture_clock_paths

Type: collection
Returns timing path collections for the capture clock. The path corresponds to the output you see from report_timing -path_type full_clock_expanded. If the capturing register is clocked by a regular clock, the attribute returns only one path in the collection. If it is clocked by a generated clock, the first path in the collection is the master clock path, followed by each dependent generated clock, until it reaches the register's clock pin.

clock_uncertainty

Type: float
Returns the clock uncertainty of the timing path. The uncertainty can be defined with the set_clock_uncertainty command.

close_edge_adjustment

Type: float
Returns the sum of the recovery amounts along the path.

common_path_pessimism

Type: float
Returns the value of the clock reconvergence common path pessimism. This attribute is defined only if you are using OCV analysis and the timing_remove_clock_reconvergence_pessimism variable is set to true.

crpr_common_point

Type: collection
Returns the clock pin corresponding to the common pin used for clock reconvergence pessimism

calculation for the path. If the launch or capture clock is ideal, the clock source is returned as the common point. If the launch clock is different from the capture clock, the attribute does not exist on the path.

depth_cell_capture

Type: float
Returns the advanced on-chip variation (AOCV) calculated depth for cells in a capture path.

depth_cell_launch

Type: float
Returns the advanced on-chip variation (AOCV) calculated depth for cells in a launch path.

depth_net_capture

Type: float
Returns the advanced on-chip variation (AOCV) calculated depth for nets in a capture path.

depth_net_launch

Type: float
Returns the advanced on-chip variation (AOCV) calculated depth for nets in a launch path.

distance_cell

Type: float
Returns the AOCV calculated distance for cells in a path.

distance_net

Type: float
Returns the AOCV calculated distance for nets in a path.

dominant_exception

Type: string
Returns the type of the dominant exception for the path: false_path, multicycle_path, or min_max_delay. This attribute exists only if the path has at least one timing exception. For more information, see the description of the -exceptions option of the report_timing command.

endpoint

Type: collection
Returns the endpoint of the timing path, for example, U1/U5/par_reg/D.

endpoint_clock

Type: collection
Returns the name of the clock at the path endpoint.

endpoint_clock_close_edge_type

Type: string
Returns the type of clock edge (rise or fall) that closes (latches) the data.

endpoint_clock_close_edge_value

Type: float
Returns the clock edge time for the endpoint.

endpoint_clock_is_inverted

Type: boolean
Returns true if the endpoint clock has been inverted.

endpoint_clock_is_propagated

Type: boolean
Returns true if the endpoint clock is a propagated clock, or false if it is an ideal clock. You can set a clock as propagated by using the set_propagated_clock command.

endpoint_clock_latency

Type: float
Returns the capture clock arrival of the timing path, excluding the endpoint clock edge time (endpoint_clock_close_edge_value attribute).

endpoint_clock_open_edge_type

Type: string
Returns the type of clock edge (rise or fall) that opens the endpoint latch. If the endpoint is edge-triggered, the open and close edges are the same.

endpoint_clock_open_edge_value

Type: float
Returns the value of the opening edge of the endpoint clock.

endpoint_clock_pin

Type: collection
Returns the complete path name of the endpoint clock pin, for example, U23/U_reg/out_reg[2]/CP.

endpoint_hold_time_value

Type: float
Returns the value of the register hold time at the timing endpoint. For example, for a flip-flop, this is the library hold time for the flip-flop cell.

endpoint_is_level_sensitive

Type: boolean
Returns true if the endpoint is a level-sensitive device, such as a latch. Returns false if the endpoint is edge-triggered.

endpoint_output_delay_value

Type: float
Returns the value of the output delay of the timing endpoint. You can set the output delay value with the set_output_delay command.

endpoint_recovery_time_value

Type: float
Returns the value of the recovery time at the timing endpoint. Recovery and removal times are often defined for the asynchronous set and clear pins of registers.

endpoint_removal_time_value

Type: float
Returns the value of the removal time at the timing endpoint. Recovery and removal times are often defined for the asynchronous set/clear pins of registers.

endpoint_setup_time_value

Type: float
Returns the value of the register setup time at the timing endpoint. For example, for a flip-flop, this is the library setup time for the flip-flop cell.

endpoint_unconstrained_reason

Type: string
Returns the reason for an unconstrained endpoint: no_capture_clock, dangling_end_point, fanin_of_disabled, no_max_check, or no_min_check. This attribute exists only if the path endpoint is unconstrained.

exception_delay

Type: float
Returns the value of the minimum or maximum delay timing exception that applies to the path, if any. If the path is not constrained by either the set_min_delay or set_max_delay command, the return value is UNINIT.

is_recalculated

Type: boolean
Returns true if the timing information for the path comes from path-based (recalculated) timing analysis.

is_recovered

Type: boolean
Returns true if one of the latches in the timing path arrived after the closing edge, and there is a violation at the latch.

launch_clock_paths

Type: collection

Returns timing path collections for the launch clock. The path corresponds to the output you see from report_timing -path_type full_clock_expanded. If the launching registers are clocked by a regular clock, the attribute returns only one path in the collection. If it is a generated clock, the first path in the collection is the master clock path, followed by each dependent generated clock, until it reaches the register's clock pin.

max_time_borrow_from_endpoint

Type: float
Returns the maximum time that could be potentially borrowed at the endpoint of the path, which is the most restrictive of the following values:

- Closing edge
- Value specified by the set_max_time_borrow command

The attribute returns

- Zero if the endpoint does not end in a latch
- Corner value of the quantity if the design uses parametric on-chip variation (POCV)

mim_pessimism

Type: float
Returns the value of the multiply instantiated module (MIM) pessimism. This attribute is defined only if you loaded the merged HyperScale binary context with MIM in the current analysis.

normalized_slack

Type: float
Returns the normalized slack of the timing path.

normalized_slack_no_close_edge_adjustment

Type: float
Returns a value equal to slack_no_close_edge_adjustment divided by the allowed propagation delay for the path. This attribute is available only if timing_enable_normalized_slack is set to true.

object_class

Type: string
Returns the class of the object, which is the string "timing_path". You cannot set this attribute.

path_group

Type: collection
Returns the path group of the timing path.

path_type

Type: string
Returns the type of timing path, either maximum or minimum. For a setup check, it is maximum. For a hold check, it is minimum.

points

Type: collection
Returns a collection of the timing points that comprise the timing path, corresponding to the timing points in the left column of a report generated by the report_timing command. A single timing path can contain many timing points. You can iterate through each timing point by using foreach_in_collection.

required

Type: float
Returns the required time at the endpoint of the timing path; excludes endpoint_clock_close_edge_value by default unless the timing_path_arrival_required_attribute_include_clock_edge variable is set to true.

session_name

Type: string
Returns the name of the saved session that the timing path belongs to. Applicable only in the interactive multi-scenario analysis (IMSA) flow.

signature

Type: string
Returns the signature of the timing path.

slack

Type: float
Returns the slack of the timing path, corresponding to the slack of a timing report. A negative value indicates a path with a timing violation.

slack_no_close_edge_adjustment

Type: float
Returns the endpoint slack of the path, without considering the recoveries along the path. The value matches the slack attribute when there are no recoveries on the path.

startpoint

Type: collection
Returns the startpoint of the timing path, corresponding to the startpoint in the header of a timing report.

startpoint_clock

Type: collection
Returns the startpoint clock name of the timing path.

startpoint_clock_is_inverted

Type: boolean
Returns true if the startpoint clock is inverted.

startpoint_clock_is_propagated

Type: boolean
Returns true if the startpoint clock is a propagated clock, or false if it is an ideal clock. You can set a clock as propagated by using the set_propagated_clock command.

startpoint_clock_latency

Type: float
Returns the launch clock arrival of the timing path, excluding the startpoint clock edge time (startpoint_clock_open_edge_value attribute).

startpoint_clock_open_edge_type

Type: string
Returns the type of clock edge (rise or fall) that launches the data.

startpoint_clock_open_edge_value

Type: float
Returns the clock edge time for the startpoint.

startpoint_input_delay_value

Type: float
Returns the value of the startpoint input delay.

startpoint_is_level_sensitive

Type: boolean
Returns true if the startpoint is a level-sensitive device such as a latch. Returns false if the startpoint is edge-triggered.

startpoint_unconstrained_reason

Type: string
Returns the reason for an unconstrained startpoint: no_launch_clock, dangling_start_point, or fanout_of_disabled. This attribute exists only if the path startpoint is unconstrained.

time_borrowed_from_endpoint

Type: float
Returns the amount of time borrowed from the timing endpoint. Time borrowing occurs in paths with level-sensitive devices.

time_lent_to_startpoint

Type: float
Returns the amount of time lent to the timing startpoint. Time borrowing occurs in paths with level-sensitive devices.

transparent_latch_paths

Type: collection
Returns the chain of "upstream" borrowing paths that lead up to the borrowing startpoint for paths with a transparent latch D-pin startpoint. To use the attribute, you must gather the path using the -trace_latch_borrow option.

variation_arrival

Type: collection
Returns the arrival time variation of the path.

variation_common_path_pessimism

Type: collection
Returns the variation of the clock reconvergence common path pessimism.

variation_endpoint_clock_latency

Type: collection
Returns the capture-clock arrival time variation.

variation_endpoint_hold_time_value

Type: collection
Returns the variation of the register hold time at the timing endpoint.

variation_endpoint_recovery_time_value

Type: collection
Returns the variation of the recovery time at the timing endpoint.

variation_endpoint_removal_time_value

Type: collection
Returns the variation of the removal time at the timing endpoint.

variation_endpoint_setup_time_value

Type: collection
Returns the variation of the register setup time at the timing endpoint.

variation_required

Type: collection
Returns the required time variation of the path.

variation_slack

Type: collection
Returns the slack variation of the path.

variation_startpoint_clock_latency

Type: collection

Returns the launch clock arrival time variation.

SEE ALSO

get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)

timing_pocvm_corner_sigma

Selects the standard deviation to be used for parametric on-chip variation analysis in PrimeTime when calculating corner values from statistical quantities.

TYPE

float

DEFAULT

3.0

DESCRIPTION

Parametric on-chip variation analysis internally computes arrival, required and slack values based on statistical distributions, When performing comparisons between these statistical quantities, PrimeTime needs to know at what corner these statistical values are evaluated for reporting to guarantee a pessimistic analysis.

The **timing_pocvm_corner_sigma** sets this corner to be used during **update_timing**.

SEE ALSO

```
timing_pocvm_corner_sigma(3)
timing_pocvm_enable_analysis(3)
```

timing_pocvm_enable_analysis

Enables graph-based parametric on-chip variation (POCV) analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Setting this variable to **true** enables parametric on-chip variation (POCV) timing analysis mode; that is, all operations in PrimeTime are performed in POCV mode. Unlike in advanced on-chip variation (AOCV) mode, there is no mode where POCV can be used in PBA only.

Note: Setting this variable to **true** automatically switches the design into **on_chip_variation** analysis mode using the **set_operating_conditions** command.

SEE ALSO

```
get_timing_paths(2)
read_aocvm(2)
report_aocvm(2)
report_timing(2)
set_operating_conditions(2)
```

timing_pocvm_enable_extended_moments

Enables extended statistical moments LVF support for parametric on-chip variation (POCV) analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Setting this variable to **true** enables extended statistical moments to be computed and propagated for timing quantities during parametric on-chip variation (POCV) timing analysis mode. That is, all operations in PrimeTime are performed in POCV mode using extended moments as described in Liberty LVF extension. The extended moments are mean_shift, std_dev and skewness as defined in Liberty LVF extension in order to model non Gaussian distribution.

Note: Setting this variable to **true** has any effect only if **timing_pocvm_enable_analysis** is also set to **true** (i.e. PrimeTime POCV analysis has been enabled as well).

SEE ALSO

`timing_pocvm_enable_analysis(3)`

timing_pocvm_enable_extrapolation_warning

Enables the printing of the warning message UITE-581 to indicate POCV sigma lookup has encountered extrapolation greater than 10 percent.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Setting this variable to **true** enables the printing of a warning message if POCV sigma lookup encounters extrapolation greater than 10 percent.

SEE ALSO

`timing_pocvm_enable_analysis(3)`

timing_pocvm_max_transition_sigma

Specifies the standard deviation to be used for parametric on-chip variation analysis in PrimeTime when reporting corner values from computed statistical transition times (for max corner analysis).

TYPE

float

DEFAULT

0.0

DESCRIPTION

Parametric on-chip variation analysis internally computes timing quantities (example arrival, required, transition times) based on statistical distributions.

When performing comparisons between statistical transition values, PrimeTime needs to know at what corner these statistical values are evaluated. This can be specified by setting the **timing_pocvm_corner_sigma** variable.

During reporting of max_transition violations using report_constraint command, a less pessimistic corner can be evaluated without performing a full timing update by selecting a corner value for reporting max_transition values only. This reporting of corner values of max_transition can be set using the **timing_pocvm_max_transition_sigma** variable.

SEE ALSO

`timing_pocvm_corner_sigma(3)`

```
timing_pocvm_report_sigma(3)
timing_enable_slew_variation(3)
timing_pocvm_enable_analysis(3)
```

timing_pocvm_precedence

Controls the precedence of POCV tables on multiple object types applying to an arc

TYPE

string

DEFAULT

file

DESCRIPTION

Set this variable to one of these values:

- **file** (the default) - POCV coefficient file takes precedence over LVF data available in library.
- **library** - POCV LVF data available in library takes precedence over coefficient file.
- **lib_cell_in_file** - the tool uses the following priority, in decreasing order of precedence, to determine the table that applies to the arc:
  - Library cell table
  - Hierarchical cell table
  - Design table

  This variable is effective only when the **timing_pocvm_enable_analysis** variable is set to **true**.

SEE ALSO

```
read_ocvm(2)
timing_pocvm_enable_analysis(3)
```

timing_pocvm_report_sigma

Specifies the standard deviation to be used for parametric on-chip variation analysis in PrimeTime when reporting corner values from statistical quantities.

TYPE

float

DEFAULT

3.0

DESCRIPTION

Parametric on-chip variation analysis internally computes arrival, required and slack values based on statistical distributions.

When performing comparisons between these statistical quantities, PrimeTime needs to know at what corner these statistical values are evaluated for reporting to guarantee a pessimistic analysis this corner is set by the **timing_pocvm_corner_sigma** variable.

During reporting, a less pessimistic corner can be evaluated without performing a full timing update by selecting a corner value for reporting only. This reporting only corner value can be set using the **timing_pocvm_report_sigma** variable. The result is guaranteed to be bounding the result one would get by setting the **timing_pocvm_corner_sigma** to the new specified corner followed by a full update_timing, but can be more pessimistic.

The value specified for **timing_pocvm_report_sigma** must be smaller than the value of the **timing_pocvm_corner_sigma** variable, or else it is ignored.

SEE ALSO

```
timing_pocvm_corner_sigma(3)
timing_pocvm_enable_analysis(3)
```

timing_point_arrival_attribute_compatibility

Includes or excludes the arrival to startpoint in the **arrival** attribute for timing points of a timing path.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Set this variable to one of the following values:

- **true** (the default) - Excludes the arrival to startpoint of timing path from the calculation of the **arrival** attribute for timing points of the timing path. The arrival to startpoint attribute includes source/propagagated clock latency(startpoint_clock_latency), input external delay (startpoint_input_delay_value), time given to startpoint(time_lent_to_startpoint) , and clock edge time. .
- **false** - Includes the arrival to startpoint of timing path in the calculation of the **arrival** attribute for timing points of the timing path. When you use this setting, the **arrival** attribute values for timing points match the arrival times reported by the **report_timing** command, except for the clock edge time (if that time is non-zero).

SEE ALSO

`report_timing(2)`

timing_path_attributes(3)

timing_point_attributes

Describes the predefined application attributes for timing_point objects.

DESCRIPTION

annotated_delay_delta

Type: float
Returns the delta delay in the timing point.

annotated_delta_transition

Type: float
Returns the delta transition in the timing point.

aocvm_coefficient

Type: float
Returns the AOCV coefficient calculated for the arc to the timing point.

applied_derate

Type: float
Returns the applied derating calculated for the arc to the timing point.

arrival

Type: float
Returns the arrival time at the timing point without accounting for the following:

- Clock latency to the startpoint clock
- Time lent to the startpoint (due to latch borrowing)
- Input delay
- The startpoint_clock_open_edge_value attribute

You need to add the preceding values to the arrival attribute value to obtain the total arrival time based on the desired startpoint.

depth

Type: float
Returns the depth for the pin or port of the timing point. This value is scaled by AOCV coefficients if they exist.

derate_factor_depth_distance

Type: float
Returns the derating factor as a function of depth and distance calculated for the arc to the timing point.

distance

Type: float
Returns the distance for the pin or port of the timing point.

guardband

Type: float
Returns the guard band calculated for the arc to the timing point.

incremental

Type: float
Returns the incremental value calculated for the arc to the timing point.

object

Type: collection
Returns the object at this point in the timing path.

object_class

Type: string
Returns the class of the object, which is the string "timing_point". You cannot set this attribute.

rise_fall

Type: string
Returns "rise" if the timing point is a rising-edge delay or "fall" if it is a falling-edge delay.

si_xtalk_bumps

Type: string
Returns the crosstalk bump at the timing point, listing each aggressor net and the voltage bumps that rising and falling aggressor transitions induce on the victim net (worst of rising and falling minimum or maximum bumps, each expressed as a decimal fraction of the rail-to-rail voltage); or gives a reason why the aggressor net has no effect on the victim net.

slack

Type: float
Returns the slack value at the timing point.

transition

Type: float
Returns the transition value at the timing point.

variation_arrival

Type: collection
Returns the arrival time variation of the timing point.

variation_increment

Type: collection
Returns the incremental variation of the timing point.

variation_slack

Type: collection
Returns the slack time variation of the timing point.

variation_transition

Type: collection
Returns the transition time variation of the timing point.

voltage

Type: float
Returns the voltage level of the timing point.

x_coordinate

Type: float
Returns the x coordinate of the physical location of the timing point.

y_coordinate

Type: float
Returns the y coordinate of the physical location of the timing point.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

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), 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 trip points and voltage level on the load cell.

No scaling is done for a post-layout flow because PrimeTime measures delays on analog waveforms.

SEE ALSO

`report_delay_calculation(2)`

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.

For example, consider a pipeline containing latches A, B, and C, clocked by clocks 1, 2, and 3, respectively. The tool 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**.

SEE ALSO

report_timing(2)
set_clock_uncertainty(2)

timing_propagate_through_non_latch_d_pin_ar

Propagates cell arcs from data pins for edge-triggered devices.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, under certain conditions, PrimeTime does not allow propagation through the cell arcs from data pins of edge-triggered devices.

To allow propagation through the cell arcs from data pins for edge-triggered devices, set this variable to **true**. Changing the setting of this variable triggers a full update timing subsequently.

SEE ALSO

`update_timing(2)`

timing_reduce_multi_drive_net_arcs

Enables or disables the collapsing of parallel timing arcs to improve PrimeTime performance and memory utilization.

TYPE

Boolean

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 mesh cell and net delays and transition times at the driver and load pins.

To improve performance and memory requirements for clock network analysis, PrimeTime has to reduce the number of timing arcs it must consider. To do this, set the **timing_reduce_multi_drive_net_arcs** variable to **true**.

The reduction operation is performed during design linking; therefore, the variable has to be set prior to that. After 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 product of the size of a net's global drivers and loads. 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 should have at most one output pin driving the mesh.

2. All driver cells must be the same type (lib_cell).
3. All driver cells relevant inputs 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.

The **PTE-046** message would specify the "selected" driver. Alternatively, obtain the selected driver by computing the back arcs to any load pin of the mesh using get_timing_arcs with the -to option.

You need to annotate accurate delays to selected mesh driver pin back arcs as well as delays from the selected driver to all the mesh load pins using **set_annotated_delay**, in addition to transition times at the selected drivers and all the load pins using **set_annotated_transition**. The **check_timing** command verifies that the reduction affected multidriven nets in the clock network and that the necessary annotated delays and transitions do exist as indicated above.

Note that the reduced cells are not physically removed from the netlist, but that no timing arcs exist from their output pins. Therefore, flows using the **write_changes** command are not be affected. However, not having the timing arcs 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.

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; therefore, 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.

SEE ALSO

```
check_timing(2)
report_delay_calculation(2)
report_timing(2)
set_annotated_delay(2)
set_annotated_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

integer

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 is not considered for reduction.

SEE ALSO

`timing_reduce_multi_drive_net_arcs(3)`

timing_remove_clock_reconvergence_pessimisr

Enables clock reconvergence pessimism removal (CRPR).

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is **true** (the default), the tool removes clock reconvergence pessimism from slack calculation and minimum pulse width checks.

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 **timing_clock_reconvergence_pessimism** variable to control CRP calculation with respect to transition sense. In the case of the capturing device being a level-sensitive latch two CRP values are calculated:

- 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 is increased by the value of crp_open and therefore 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.

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 **si_enable analysis** variable is set to **true**, delays in the clock network could also include delta delays resulting from crosstalk interaction. Such delays are dynamic in nature, that is, they can 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.

PrimeTime SI considers delta delays as part of the CRP calculation only if the type of timing check deployed derives its data from the same clock cycle.

Similarly, if dynamic annotations have been set on the design, the clock delays computed using these annotations are only 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 can be specified with the **set_clock_latency** command, or dynamic rail voltage, which can be specified with the **set_rail_voltage** command.

In transparent-latch based designs, you should set the **timing_early_launch_at_borrowing_latches** variable to **false** when clock reconvergence pessimism removal (CRPR) is enabled. In this case, CRPR applies even to paths whose startpoints are borrowing, leading to better pessimism reduction overall.

Any effective change in the setting 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.

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, set the **timing_crpr_remove_clock_to_data_crp** variable to **true**.

To turn CRPR on:

```
pt_shell> set_app_var 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_crpr(2)
report_min_pulse_width(2)
report_timing(2)
set_clock_latency(2)
set_rail_voltage(2)
si_enable_analysis(3)
timing_clock_reconvergence_pessimism(3)
timing_crpr_remove_clock_to_data_crp(3)
timing_crpr_threshold_ps(3)
timing_early_launch_at_borrowing_latches(3)

timing_report_always_use_valid_start_end_poi

Requires the **-from**, **-rise_from**, and **-fall_from** options to specify valid timing startpoints and the **-to**, **-rise_to**, and **-fall_to** options to specify valid timing endpoints.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable affects the way the **report_timing**, **report_bottleneck**, and **get_timing_path** commands interpret the **-from**, **-rise_from**, **-fall_from**, **-to**, **-rise_to**, and **-fall_to** options.

When set to **false** (the default), the *from_list* objects are interpreted as all pins found by given objects. Objects specified with an asterisk (*) wildcard or cell name might return invalid startpoints or endpoints. For example, **-from [get_pins FF/*]** includes input, output, and asynchronous pins. The tool considers these invalid startpoints or endpoints to be throughpoints and continues the path searching. Although it is convenient to use, it is not suggested because of longer runtimes.

To report only valid startpoints and endpoints, set this variable to **true**. It is always suggested to use input ports or register clock pins for the *from_list* objects and output ports or register data pins for the *to_list* objects.

SEE ALSO

```
get_timing_paths(2)
report_bottleneck(2)
report_timing(2)
```

timing_report_fixed_width_columns_on_left

Controls the position where the instance/net/pin name details are displayed during the timing report process.

TYPE

BOOLEAN

DEFAULT

FALSE

DESCRIPTION

Controls the position where the pin name details are displayed when you use the **report_timing** command. Valid values are **false** (the default), and **true**.

Full hierarchical pin names are becoming very long thereby affecting the readability of the generated timing reports. Columns can become misaligned. To help remedy this situation, this variable displays the instance/net/pin name (ie the Point column in the timing report) at the end of the report. This variable will make sure that all the columns are intact.

In the distributed multi-scenario analysis (DMSA) flow, to enable this feature,set this variable to TRUE in the master. The tool uses the variable setting in the master regardless of the variable setting in the slaves.

EXAMPLES

The following example shows a conventional report and a report with the numeric, fixed-width columns on the left.

```
pt_shell> report_timing
...
  Startpoint: SET (input port clocked by CLK1)
  Endpoint: n02_reg (rising edge-triggered flip-flop clocked by CLK1)
  Path Group: CLK1
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock CLK1 (rise edge)                   0.00       0.00
  clock network delay (propagated)         0.00       0.00
  input external delay                     0.50       0.50 f
  SET (in)                                 0.00       0.50 f
  n01_reg/Q (FDES)                         0.60 H     1.10 r
  OR2S_cell/Y (OR2S)                       0.00       1.10 r
  n02_reg/D (FDES) <-                      0.00       1.10 r
  data arrival time                                   1.10

  clock CLK1 (rise edge)                  10.00      10.00
  clock network delay (propagated)         0.00      10.00
  clock reconvergence pessimism            0.00      10.00
  n02_reg/T (FDES)                                   10.00 r
  library setup time                      -0.35       9.65
  data required time                                  9.65
  -----------------------------------------------------------
  data required time                                  9.65
  data arrival time                                  -1.10
  -----------------------------------------------------------
  slack (MET)                                         8.55

pt_shell> set_app_var timing_report_fixed_width_columns_on_left true
true
pt_shell> report_timing
...
  Startpoint: SET (input port clocked by CLK1)
  Endpoint: n02_reg (rising edge-triggered flip-flop clocked by CLK1)
  Path Group: CLK1
  Path Type: max

      Incr       Path    Point
  -------------------------------------------------------
      0.00       0.00    clock CLK1 (rise edge)
      0.00       0.00    clock network delay (propagated)
      0.50       0.50 f  input external delay
      0.00       0.50 f  SET (in)
      0.60 H     1.10 r  n01_reg/Q (FDES)
      0.00       1.10 r  OR2S_cell/Y (OR2S)
      0.00       1.10 r  n02_reg/D (FDES) <-
                 1.10    data arrival time

     10.00      10.00    clock CLK1 (rise edge)
      0.00      10.00    clock network delay (propagated)
      0.00      10.00    clock reconvergence pessimism
                10.00 r  n02_reg/T (FDES)
     -0.35       9.65    library setup time
                 9.65    data required time
  -------------------------------------------------------
                 9.65    data required time
                -1.10    data arrival time
  -------------------------------------------------------
                 8.55    slack (MET)
```

SEE ALSO

report_timing(2)

timing_report_hier_stub_pin_paths

Specifies whether the **report_timing** command reports paths ending at HyperScale stub pins (internal endpoints) in the internal separate path group.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

A timing path ending at a stub pin is a HyperScale-specific behavior. In the HyperScale flow, there are block-level and top-level analyses. Stub pin paths exist at the top-level analysis and are internal endpoints within a HyperScale block. Stub pins exist because block internal register-to-register paths are not fully retained at the top level. A stub pin is created when a starting register has paths leading to block output ports -- that is, starting from an interface register -- while the starting register also has timing paths leading to other registers inside the block, which makes the path shared with block internal paths. The points where a path branches to interface paths and internal paths are designated as stub pins. When a pin becomes a stub, its entire transitive fanout, including all the logic on the paths from the stub pin to the capture registers in the block, plus the clock paths of these internal capture register, are not retained in the block abstraction and therefore invisible at the top-level analysis. A stub pin is often one of the load pins of a multiload net. Topologically speaking, a stub pin is symmetric to a side input, which happens on a block input interface due to multifanin cells.

To reproduce the worst slack on the interface register in the transitive fanin of the stub pins, HyperScale can automatically capture the required times during block-level analysis and annotate on stub pins during top-level analysis when block abstraction is reused. These required times do not change the timing analysis on the fanin, but produced pin slack as if the fanout still presents, which is very important for ECO.

Set the **timing_report_hier_stub_pin_paths** variable to one of these values:

- **true** - Reports the timing paths ending at stub pins in the **HyperScale_stub_default** intrinsic path group. Note that these stub pins are usually not timing path endpoints in block-level or top-level flat analysis.
- **false** - Omits reporting the paths ending at stub pins.

Note:
This variable does not affect timing analysis; it only affects how the **report_timing** and **get_timing_paths** commands report timing paths.

EXAMPLES

The following example shows the path reported in the **HyperScale_stub_default** intrinsic path group when the **timing_report_hier_stub_pin_paths** variable is set to **true**.

```
report_timing -path_type full_clock_expanded -from block/ffa2/Q
****************************************
Report : timing
        -path_type full_clock_expanded
        -delay_type max
        -max_paths 1
        -sort_by group
Design : SIMPLE
 ...
****************************************

  Startpoint: block/ffa2 (rising edge-triggered flip-flop clocked by SYSCLK)
  Endpoint: block/ffa4/D
              (internal path endpoint clocked by SYSCLK)
  Path Group: **HyperScale_stub_default**
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock SYSCLK (rise edge)                0.000      0.000
  clock source latency                    0.700      0.700
  CLK (in)                                0.000      0.700 r
  block/clk (BLOCK)                       0.000      0.700 r
  block/cbufa1/Y (CLKBUFX2)               0.385      1.085 r
  block/ffa2/CK (DFFX2)                   0.000      1.085 r
  block/ffa2/Q (DFFX2) <-                 0.460      1.545 f
  block/ffa4/D (DFFX2)                    0.000      1.545 f
  data arrival time                                  1.545

  clock SYSCLK (rise edge)                4.000      4.000
  clock source latency                    0.400      4.400
  CLK (in)                                0.000      4.400 r
  block/clk (BLOCK)                       0.000      4.400 r
  block/cbufa1/A (CLKBUFX2)               0.000      4.400 r
  output external delay                  -0.004      4.396
  data required time                                 4.396
  -----------------------------------------------------------
  data required time                                 4.396
  data arrival time                                 -1.545
  -----------------------------------------------------------
  slack (MET)                                        2.850
```

```
1

report_path_group
****************************************
Report : path_group
Design : SIMPLE
 ...
****************************************

Path_Group     Weight   From      Through        To
------------------------------------------------------------
**HyperScale_stub_default**
                1.00   -         -              -
**async_default**
                1.00   -         -              -
**clock_gating_default**
                1.00   -         -              -
**default**     1.00   -         -              -
SYSCLK          1.00   *         *              SYSCLK

1
```

SEE ALSO

get_timing_paths(2)
report_timing(2)
hier_enable_analysis(3)

timing_report_include_eco_attributes

Includes cell and net attribute information in reports generated by the **report_timing** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to true, a column labeled "Attributes" is added in the output of the **report_timing** command to show information about cells and nets in the timing report:

- **dont_touch** (for cell or net)
- **dont_use** (for cell)
- **ideal_net** (for net)
- **via_ladder** (for pin)

These letters d, i, and u appear in the Attributes column to indicate where the attribute is true for a cell or net in the timing point list:

```
pt_shell> set_app_var timing_report_include_eco_attributes true
 ...
pt_shell> report_timing ...
 ...
Attributes:
    d - dont_touch
    u - dont_use
    i - ideal_net
    V - via_ladder
```

```
Startpoint: A1/B1/reg_out
              (rising edge-triggered flip-flop clocked by clk)
  Endpoint: reg_out[11]
              (output port clocked by clk)
  Path Group: in2out
  Path Type: max

  Point                                    Incr       Path     Attributes
  ----------------------------------------------------------------------------
  clock clk (rise edge)                    0.00       0.00
  clock network delay (propagated)         3.30       3.30
  A1/B1/reg_out_reg_11_/CP (dfn)           0.00       3.30 r          V
  A1/B1/reg_out_reg_11_/Q (dfn)            0.87       4.18 f   i      V
  icc_route_opt6/ZN (inv7)                 0.00       4.18 r   i d u
  icc_route_opt8/ZN (inv7)                 0.07       4.25 f     d u
  reg_out[11] (out)                        0.01       4.25 f
  data arrival time                                   4.25

  clock clk (rise edge)                    1.67       1.67
  clock network delay (propagated)         0.00       1.67
  clock reconvergence pessimism            0.00       1.67
  output external delay                   -0.17       1.50
  data required time                                  1.50
  ----------------------------------------------------------------------------
  data required time                                  1.50
  data arrival time                                  -4.25
  ----------------------------------------------------------------------------
  slack (VIOLATED)                                   -2.75
```

When the variable is set to false (the default), the Attributes column is not displayed in timing reports.

SEE ALSO

report_timing(2)

timing_report_invert_clock_min_pulse_width_c

Determines how the high/low min pulse width constraints set on a clock are applied at a pin.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable determines how the high/low min pulse width constraints set on a clock are applied at a pin.

When this variable is **false**, the high/low min pulse width constraint, set by **set_min_pulse_width** on a clock, to be applied at a given pin in the clock network is selected independently of the sense of the clock network from clock source to the pin.

When this variable is **true**, the high/low min pulse width constraint, set by **set_min_pulse_width** on a clock, to be applied at a given pin in the clock network is selected based on the sense of the clock network from clock source to the pin. This means that if the clock waveform is inverted at pin relative to the clock source then the min pulse width constraints are also inverted. For non unate networks, the most restrictive of high/low min pulse width constraints is used.

Note that this variable only impacts min pulse width constraints set on clocks. Constraints set on design/pin with **set_min_pulse_width** or inferred from the library are applied directly as specified.

SEE ALSO

```
report_constraint(2)
report_min_pulse_width(2)
set_min_pulse_width(2)
```

timing_report_output_delay_on_clock_source

Recognizes output delays specified on clock sources defined on output ports.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, the tool will recognize the output delays specified on clock sources defined on output ports. As a result, the tool will be able to set output path delays and therefore, the output paths will be reported.

By default, the variable is set to **false** and the tool discards the output delays specified on clock source pins defined on output ports.

SEE ALSO

`set_output_delay(2)`

timing_report_pin_names_in_header

Causes the startpoint and endpoint lines of a timing report header to display specific pins instead of only the cells.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable affects how the **report_timing** command displays the startpoint and endpoint at the beginning of the path report when the object is a cell input or output. When the variable is set to **false** (the default), the cell instance names are displayed:

```
Startpoint: ORCA/BLENDER/s3_reg[18]
             (rising edge-triggered flip-flop clocked by SCLK)
Endpoint: ORCA/BLENDER/s4_reg[31]
             (rising edge-triggered flip-flop clocked by SCLK)
```

When the variable is set to **true**, the specific startpoint and endpoint pins are displayed:

```
Startpoint: ORCA/BLENDER/s3_reg[18]/CP
             (rising edge-triggered flip-flop clocked by SCLK)
Endpoint: ORCA/BLENDER/s4_reg[31]/D
             (rising edge-triggered flip-flop clocked by SCLK)
```

This variable also affects reports generated by other commands that display timing paths, such as the verbose modes of **report_constraint** and **report_clock_timing**.

SEE ALSO

```
report_timing(2)
report_clock_timing(2)
report_constraint(2)
```

timing_report_recalculation_status

Displays 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 setting of the variable, as follows:

- **none** - Displays no messages.
- **low** - Displays a message only at the end of the recalculation for each clock group.
- **medium** - Displays messages only at the beginning and end of the recalculation for each clock group.
- **high** - Displays all messages for **medium**; in addition, the tool displays the total number of endpoints to search and the completion percentage for searching these endpoints.

The following example shows information messages 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 maximum number of paths recalculated at an endpoint was 10.
```

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
```

timing_report_skip_early_paths_at_intermediat

Disables the reporting of timing paths with early arrival at intermediate latches.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

A path is said to arrive early at an intermediate latch if it arrives before the clock causes the latch to open.

To control the reporting of timing paths that arrive early at intermediate latches, set this variable to one of these values:

- **false** (the default) - The **report_timing** command reports paths that arrive early at intermediate latches. These paths are indicated as "(early)" in the report at the individual transparency window where the arrival arrived early. The tool tries to identify which window is early on the path. Due to numerical rounding on paths that are only slightly early, it is possible for some early paths to be filtered out.
- **true** - The **report_timing** command skips paths that arrive early at intermediate latches.

This variable is effective only if you enable advanced analysis through transparent latches by setting the **timing_enable_through_paths** variable to **true**). With advanced analysis through transparent latches, you can report paths through latches as a single timing path that is composed of a sequence of paths segments. Each segment terminates at an intermediate latch.

SEE ALSO

```
report_timing(2)
timing_enable_through_paths(3)
```

timing_report_status_level

Controls the number of progress messages displayed during the timing report process.

TYPE

string

DEFAULT

none

DESCRIPTION

Controls the number of progress messages displayed when you use the **report_timing** and **report_constraint** commands. Valid values are **none** (the default), **low**, **medium**, and **high**.

The number of messages varies based on the setting of the variable, as follows:

- **none** - Displays no messages.
- **low** - Displays messages only at the beginning and end of the timing report.
- **medium** - Displays all messages for **low**; in addition, displays messages at the beginning of searching for each clock group.
- **high** - Displays all messages for **medium** when you use the **report_timing** command; in addition, displays the total number of endpoints to search and the completion percentage for searching these endpoints.

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, set the **timing_update_status_level** variable.

SEE ALSO

report_constraint(2)
report_timing(2)
timing_update_status_level(3)

timing_report_trace_ideal_clocks

Causes clock paths to be shown for ideal clock networks in **report_timing -path_type full_clock_expanded** and associated commands that display clock paths.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables the display of ideal clock network paths by **report_timing -path_type full_clock_expanded** and similar reporting commands (**report_clock_timing -verbose** and **report_min_pulse_width -path_type full_clock_expanded**).

When the variable is set to **false** (the default), the clock path through an ideal clock network is not displayed. Instead, a single line summarizes the ideal clock latency. The topological path may nonetheless be of interest. For example, you might want to determine which ideal clock path is reaching a register where you did not expect it.

When the variable is set to **true**, the paths through ideal clock networks are displayed by **report_timing -path_type full_clock_expanded**, even though the delays along such paths do not impact the timing of associated registers. Timing calculations remain governed by the ideal settings.

Note that the path increments shown in the ideal network might not match the increments that appear when the ideal network is changed to propagated by the **set_propagated_clock** command.

When the variable is set to **true**, a timing path gathered using the **get_timing_paths** command supports the query of **capture_clock_paths** and **launch_clock_paths** attributes even when the path is clocked by ideal clocks.

SEE ALSO

report_timing(2)
get_timing_paths(2)
report_clock_timing(2)
report_ideal_network(2)
set_ideal_latency(2)
set_ideal_network(2)
set_propagated_clock(2)

timing_report_unconstrained_paths

Specifies whether the tool searches for unconstrained paths 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. If the design has many unconstrained paths, this setting uses less runtime because the tool does not spend time analyzing the unconstrained paths.

To search for and report unconstrained paths as well as constrained paths, set this variable to **true**. Unconstrained paths have a startpoint or endpoint without a defined clock. For example, if no clocks, input delays, or output delays are defined, all paths are unconstrained. The tool first searches for constrained paths that meet the search criteria; if there are none, it then searches for unconstrained paths.

When searching for unconstrained paths, instead of finding the paths with the least slack, the **get_timing_paths** and **report_timing** commands look for and report paths with the longest delay by default, or for paths with the shortest delay if the **-delay_type** option is set to **min**. Searching for unconstrained paths can cause unexpectedly long runtimes, as reported by the UITE-413 warning message.

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
```

timing_report_union_tns

Specifies whether to use the union method for reporting total negative slack and number of violating endpoints.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

If this variable is set to **false**, **report_qor -summary** computes the design's TNS as the sum of TNS for all path groups in that design. In multi-scenario analysis, the design's TNS is computed as the sum of TNS for all path groups in all scenarios in focus.

If the variable is set to **true** (the default), **report_qor -summary** counts each violating endpoint only once when computing the total negative slack (TNS) and the number of violating endpoints for the design. In multi-scenario analysis, the design's TNS is computed as the sum of worst negative violations per endpoint across all scenarios and path groups.

SEE ALSO

`report_qor(2)`

timing_report_use_worst_parallel_cell_arc

Enables uniquification of paths through parallel cell arcs.

TYPE

Boolean

DEFAULT

true

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. Use this variable to specify whether to report every path through parallel cell arcs.

When you set this variable to **false**, PrimeTime reports all paths through parallel cell arcs.

When you set this variable to **true**, PrimeTime reports only the worst path through a set of parallel cell arcs. PrimeTime chooses the arc with the worst delay to determine the worst path. This variable setting has no effect in designs with no parallel cell arcs.

SEE ALSO

```
get_timing_paths(2)
report_timing(2)
```

timing_save_block_level_reporting_data

Forces PrimeTime HyperScale to save all report data at the block level.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls how block-level data is generated in the HyperScale flow for reporting commands. By default, the reporting command has to be called at the block level in order for the data to be available at the top level. If this variable is set to **true**, the generation of that data is done automatically at the block level. There is no need for an explicit command call at the block level to get block data at the top.

SEE ALSO

```
report_analysis_coverage(2)
report_global_timing(2)
report_qor(2)
```

timing_save_hier_context_data

Specifies whether the **write_hier_data** command writes out the hierarchical context data for HyperScale blocks.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

In the HyperScale analysis flow, the **set_hier_config** command specifies which blocks in the design are HyperScale blocks. By default, when the use the **set_hier_config** command, the design is considered a top-level design, and the **write_hier_data** command writes out the hierarchical context data for HyperScale blocks in the design.

In some flows, including some bottom-up flows, you do not need to write out the block context for the HyperScale blocks in the design and you do not need to perform hierarchical scope checking for those blocks. In that case, to save runtime and disk space, set the **timing_save_hier_context_data** variable to **false**. Then the **write_hier_data** command skips all context capture and scope checking, except a small subset needed for nonfixable scope violations. Nonfixable violations are still checked and reported by the **report_constraint** command.

SEE ALSO

`report_constraint(2)`

```
set_hier_config(2)
write_hier_data(2)
timing_save_hier_model_data(3)
```

timing_save_hier_model_data

Specifies explicitly that the current design is a HyperScale block-level design, which ensures that the **write_hier_data** command writes out a HyperScale block model for the design.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

In the HyperScale analysis flow, the current design is treated as a block-level design under any one or more of the following conditions:

- The design does not contain HyperScale blocks (the **set_hier_config** command is not used in the session).
- The **read_context** command is used in the session.
- The **timing_save_hier_model_data** variable is set to **true**.

In any of these cases, the **write_hier_data** command writes out a HyperScale block model for the current design. This model can be used for analysis at the next higher level of hierarchy.

In most block-level designs, it is not necessary to set the **timing_save_hier_context_data** variable because the lack of lower-level HyperScale blocks, or using the **read_context** command, already implies that it is a block-level design.

However, in the unusual case of a mid-level HyperScale block containing lower-level HyperScale blocks, when you do not use the **read_context** command, set the **timing_save_hier_context_data** variable to **true** to ensure that the **write_hier_data** command writes out a HyperScale block model for the

design.

SEE ALSO

```
read_context(2)
set_hier_config(2)
write_hier_data(2)
timing_save_hier_context_data(3)
```

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 set to **true**, the arrival and required times of all pins of the design are kept in memory. When set to **false** (the default), arrival and required times are stored on an as-needed basis for the analysis you are performing.

This variable is very similar in effect to the **timing_save_pin_arrival_and_slack** variable, enabling the same features (particularly slack and arrival window attribute query). To query slack attributes or arrival window attributes on pins that are not endpoints of the design, set one or both of these variables to **true**.

You should also set this variable to **true** in the specific case where the **write_sdf_constraints** command forms part of your flow, as this command requires additional information be stored at all pins. If the **write_sdf_constraints** command is used while this variable is set to **false**, it is set to **true** automatically and an informational message issued.

SEE ALSO

`report_timing(2)`

```
write_sdf_constraints(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 set to **true**, the arrival times and slacks of all pins of the design are kept in memory. When set to **false** (the default), arrival times and slacks are preserved only for endpoints of the design.

This variable is very similar in effect to the **timing_save_pin_arrival_and_required** variable, enabling the same features (particularly slack and arrival window attribute query). To query slack attributes or arrival window attributes on pins that are not endpoints of the design, set one or both of these variables to **true**.

SEE ALSO

```
report_timing(2)
timing_save_pin_arrival_and_required(3)
```

timing_separate_hier_side_inputs

Specifies if PrimeTime HyperScale analysis should group paths starting from side inputs (internal startpoint) into its own and separate path groups.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Side input path is a HyperScale specific behavior. In the HyperScale analysis flow, there are block-level and top-level analyses. Side input paths at block level are internal register to register paths, and they converge with interface paths by sharing some common logic through some multiple-input cells, For example, an AND gate whose A pin is on an interface path starting from a block port and leading to some register, while its B pin is on a path starting from a register cell. During HyperScale block data reduction, paths through the A pin are retained while the fanin of B pin is removed. This pin is the side input pin of the interface path. To have the same accuracy of flat analysis at the HyperScale top level for the paths leading to the common register of the AND gate, the tool captures and annotates all the timing data propagated at B pin in binary format. At the top level, the paths appear to be starting from these dangling side input pins.

When this variable is set to **false** (the default), the paths starting from these side inputs maintain their original path group (capture clock by default). **report_timing** and **get_timing_paths** commands report them as critical paths in the same path group as the interface paths converging at the same block interface registers.

When this variable is set to **true**. An internal path group "**HyperScale_side_inputs**" is automatically created and all the paths starting from the block side inputs at HyperScale top level analysis are analyzed and reported in this group. Note this does not impact the actual analysis accuracy or QoR, affects only the

path reporting.

EXAMPLES

The following example shows the path reported in HyperScale side input when the variable is set to **true**.

```
report_timing -from $side_input
****************************************
Report : timing
        -path_type full
        -delay_type max
        -max_paths 1
 ...
****************************************

  Startpoint: blk/u3/B (internal path startpoint clocked by top_CLK)
  Endpoint: blk/ff1 (rising edge-triggered flip-flop clocked by top_CLK)
  Path Group: **HyperScale_side_input**
  Path Type: max

  Point                                    Incr       Path
  -----------------------------------------------------------
  clock top_CLK (rise edge)                0.00       0.00
  clock network delay (propagated)         0.02       0.02
  input external delay                     4.68       4.70 f
  blk/u3/B (ND2)                           0.00       4.70 f
  blk/u3/Z (ND2)                           0.64 &     5.34 r
  blk/ff1/D (FD2)                          0.00 &     5.34 r
  data arrival time                                   5.34

  clock top_CLK (rise edge)               10.00      10.00
  clock network delay (propagated)         1.16      11.16
  blk/ff1/CP (FD2)                                   11.16 r
  library setup time                      -0.85      10.31
  data required time                                 10.31
  -----------------------------------------------------------
  data required time                                 10.31
  data arrival time                                  -5.34
  -----------------------------------------------------------
  slack (MET)                                         4.97
1

report_path_group
****************************************
Report : path_group
 ...
****************************************

Path_Group     Weight   From      Through        To
-------------------------------------------------------------------------------
**HyperScale_side_input**
                1.00   { blk/u3/B ... }
                                 *              *
**async_default**
                1.00   -         -              -
**clock_gating_default**
                1.00   -         -              -
**default**     1.00   -         -              -
```

```
    top_CLK          1.00   *         *              top_CLK
    top_CLK2         1.00   *         *              top_CLK2
```

SEE ALSO

```
get_timing_paths(2)
hier_enable_analysis(3)
report_timing(2)
```

timing_simultaneous_clock_data_port_compati

Enables or disables the simultaneous behavior of input port as a clock and data port.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When the **timing_simultaneous_clock_data_port_compatibility** variable is set to **false** (the default), an input port can behave simultaneously as a clock and data port, 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, the tool considers the port exclusively as a clock port and imposes restriction on the data edges that are launched. Also, you cannot set input delays relative to another clock.

SEE ALSO

```
set_clock_latency(2)
set_input_delay(2)
```

timing_single_edge_clock_gating_backward_co

Enables clock gating checks when only one edge of a clock arrives at the clock pin of a clock gate.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **false** (the default), the tool skips clock gating checks for the missing edge. The clock gating check is performed for only the edge that is not missing. The PTE-109 warning message indicates which edge is missing and which check is not performed.

When this variable is set to **true**, the tool skips clock gating checks if only one edge of the clock (rise or fall) arrives at the clock gate. When this happens, the PTE-074 warning message is issued.

SEE ALSO

PTE-074(n)
PTE-109(n)

timing_slew_threshold_scaling_for_max_transi

Specifies the compatibility mode for timing slew threshold scaling.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the scaling of transition time for slew threshold is enabled.

To disable the scaling of transition time for slew threshold, set this variable to **true**. This was the default behavior before PrimeTime version Z-2006.12.

SEE ALSO

`set_max_transition(2)`

timing_through_path_max_segments

Specifies the maximum number of latches for reporting paths through latches.

TYPE

integer

DEFAULT

5

DESCRIPTION

When the **timing_through_path_max_segments** variable is set to a nonzero value, the tool selects some latch data pins on paths with many transparent latches to behave as they would if **timing_enable_through_paths** were **false**; this limits the reporting on the long path but speeds up analysis. With a small nonzero value, the tool selects many transparent latch data pins in the design. With a larger value, the tool selects fewer latch data pins.

When you set **timing_through_path_max_segments** to 0, the tool selects no latch data pins. Reporting on paths through many latches is allowed, but analysis might be slower.

The **timing_through_path_max_segments** variable has no effect if the **timing_enable_through_paths** variable is set to **false**.

SEE ALSO

```
report_timing(2)
timing_enable_through_paths(3)
```

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

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, the **update_timing** command is fast); however, 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, the **update_timing** command is fast); however, 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 to the **low** value. However, even with this small 10% bound, PrimeTime can accommodate a relatively large number of changes. Therefore, it is unlikely that you need to change this default setting.
- **high**: The computational effort is high (that is, the **update_timing** command becomes slow); however, there is no increase in the memory used for the initial timing of the design.

If a design is timed again 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 **update_timing** command, and the capacitance on a port was changed using the **set_load** command, the effort spent in the execution of a

subsequently issued the **update_timing** command is smaller than that for the first issued the **update_timing** command.

SEE ALSO

set_load(2)
update_timing(2)

timing_update_status_level

Controls the number of progress messages displayed during the timing update process.

TYPE

string

DEFAULT

none

DESCRIPTION

This variable controls the number of progress messages displayed during the timing update process. The following settings are allowed:

- **none** - Displays no messages.
- **low** - Displays messages only at the beginning and the end of the update.
- **medium** - Displays all messages for the **low** setting, with the addition of messages for intermediate timing update steps, constant propagation, delay calculation, and slack computation.
- **high** - Displays all messages for the **medium** setting, with the addition of messages for the delay calculation and arrival calculation steps, the completion percentage in large designs, and groups for which slack computation is performed.

SEE ALSO

```
report_timing(2)
update_timing(2)
```

timing_use_constraint_derates_for_pulse_chec

Enables or disables using timing constraint derates for min_pulse_width and min_period constraints.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If you set this variable to **true**, PrimeTime uses factors defined by the **set_timing_derate -cell_check -late** command to derate the min_pulse_width and min_period constraints as follows:

- **set_timing_derate -cell_check -late -fall** applies to min_pulse_width checks for low pulses.
- **set_timing_derate -cell_check -late -rise** applies to min_pulse_width checks for high pulses.
- Greater of **set_timing_derate -cell_check -late -rise** and **set_timing_derate -cell_check -late -fall** applies to min_period checks.

SEE ALSO

`report_min_pulse_width(2)`
`set_timing_derate(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 you to sacrifice slew calculation for performance in an SDF flow. You can set this variable to one of these values:

- **auto** (the default) - Allows automatic switching to the SDF flow if more than 95 percent of delay arcs on a design have annotated values.
- **always** - Uses a zero value for transition time on the load pins of fully annotated delay arcs. Fully annotated arcs have values for both rise and fall, either read from an SDF file, or set with the **set_annotated_delay** command. If blocks of arcs that are not annotated exist, delay is estimated using the best available slew at the inputs.
- **never** - Does not use the SDF flow.

SEE ALSO

`read_sdf(2)`

training_data_directory

Enables Machine learning for PrimeTime's **fix_eco_power** command when a directory is specified.

TYPE

string

DEFAULT

An empty string

DESCRIPTION

When this variable is set to a string that specifies a directory name, machine learning is enabled for the **fix_eco_power** command.

If this variable is set to a null string, the **fix_eco_power** command runs in a normal mode.

For distributed multi-scenario analysis (DMSA), set this variable in the master.

Once the training data directory is specified, PrimeTime attempts to load training data for power optimization as well as write new training data after PrimeTime learns new optimization moves.

For further details about power optimization with machine learning, see the man pages of the **fix_eco_power** command.

SEE ALSO

`fix_eco_power(2)`

upf_allow_DD_primary_with_supply_sets

Enables use of a domain-dependent primary with an explicit supply set.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable coexisting of domain-dependent primary supply nets with explicit supply sets.

SEE ALSO

`load_upf(2)`

upf_attributes

Describes the predefined application attributes for IEEE 1801, also known as Unified Power Format (UPF).

DESCRIPTION

UPF-related attributes exist on the following object classes:

- upf_power_domain
- upf_power_switch
- upf_supply_net
- upf_supply_port
- upf_supply_set

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
upf_power_domain_attributes(3)
upf_power_switch_attributes(3)
upf_supply_net_attributes(3)
upf_supply_port_attributes(3)
upf_supply_set_attributes(3)
```

upf_create_implicit_supply_sets

Enables creation of supply set handles for the power domains.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Set this variable to **true** to enable creation of supply set handles. When this variable is **true**, the tool creates the **primary**, **default_isolation**, and **default_retention** supply set handles while creating the power domains. When this variable is **false**, supply set handles are not created for any power domains.

Note:
Set this variable before creating the power domains. After creating the power domains, this variable is considered read-only. The tool issues an error if you change the value of this variable after creating the power domain.

SEE ALSO

`create_power_domain(2)`

upf_enable_pst

Enables or disables UPF PST constructs in the tool.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable needs to be set to **true** before load_upf to enable UPF PST constructs. With UPF PST, PrimePower can perform PST based power analysis in averaged power analysis mode. User can then use **report_power -pst** to show power consumption per valid power state combination at the top level of the design.

SEE ALSO

`add_power_state(2)`
`report_power(2)`

upf_name_map

Specifies a list of {design_name name_map_file} to be used during golden upf reapplication.

TYPE

list

DEFAULT

NULL

DESCRIPTION

This variable specifies a list in the format of {*design_name name_map_file*} to be used when golden UPF is reapplied to a post synthesis netlist. The *name_map_file* is the name map file for the *design_name* design. The specified *name_map_file* guides the golden UPF reapplication on the *design_name* design.

SEE ALSO

```
load_upf(2)
enable_golden_upf(3)
```

upf_power_domain_attributes

Describes the predefined application attributes for upf_power_domain objects.

DESCRIPTION

default_isolation_supply

Type: collection
Returns the UPF default isolation supply set handle associated with the power domain.

default_retention_supply

Type: collection
Returns the UPF default retention supply set handle associated with the power domain.

full_name

Type: string
Returns the hierarchical name of the UPF power domain.

name

Type: string
Returns the name of the UPF power domain.

primary_supply

Type: collection
Returns the UPF primary supply set handle associated with the power domain.

scope

Type: string
Returns the scope of the power domain, which is the level of logic hierarchy designated as the root of the domain.

supplies

Type: string
Returns the function keyword and supply set name for each supply of the power domain, for example, {primary TOP.primary} {default_retention TOP.default_retention} {default_isolation TOP.default_isolation}

SEE ALSO

get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
upf_attributes(3)

upf_power_switch_attributes

Describes the predefined application attributes for upf_power_switch objects.

DESCRIPTION

control_net

Type: collection
Returns the control net.

control_port_name

Type: string
Returns the control port name.

domain

Type: collection
Returns the domain.

full_name

Type: string
Returns the hierarchical name of the UPF power switch.

input_supply_net

Type: collection
Returns the input supply net.

input_supply_port_name

Type: string
Returns the name of the input supply port.

name

Type: string
Returns the name of the UPF power switch.

on_state

Type: string
Returns a list in the following format: state_name input_supply_port boolean function.

output_supply_net

Type: collection
Returns the output supply net.

output_supply_port_name

Type: string
Returns the name of the output supply port.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
upf_attributes(3)
```

upf_power_switches_always_on

Controls if UPF power switches are considered to be always on when tracing pg connectivty.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Set this variable to **true** to enable tracing of the pg network through UPF power switches. As a result the supply_nets on both sides of the switch will be considered to be always at the same voltage. When this variable is set to **true** the **set_voltage** and **set_voltage_levels** will apply to all the connected suply_net segments on both sides of the switches. Also for SMVA analysis there will be no domain crossing inferred between supply_nets on the two sides of the switches. The behavior of the **upf_power_switches_always_on** variable is analogous to the connections inferred between supply nets declared as equivalent by the **set_equivalent** command.

Note:
Set this variable before assigning voltages or supplies. After creating the power domains, this variable is considered read-only. The tool issues an error if you change the value of this variable after creating power domains.

SEE ALSO

`set_voltage(2)`

```
    set_voltage_levels(3)
    set_equivalent(3)
```

upf_supply_net_attributes

Describes the predefined application attributes for upf_supply_net objects.

DESCRIPTION

domains

Type: collection
Returns the domains of the supply net.

full_name

Type: string
Returns the hierarchical name of the supply net.

name

Type: string
Returns the simple name of the supply net.

resolve

Type: string
Returns the resolution when the UPF supply net is driven by multiple power switches: unresolved, parallel, one_hot, or parallel_one_hot.

static_prob

Type: float
Returns the probability of logic 1 (on) for power analysis.

supply_group

Type: collection
Returns the supply group to which the supply net belongs.

voltage_max

Type: float
Returns the maximum-delay operating voltage of the supply net.

voltage_min

Type: float
Returns the minimum-delay operating voltage of the supply net.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
upf_attributes(3)
```

upf_supply_port_attributes

Describes the predefined application attributes for upf_supply_port objects.

DESCRIPTION

direction

Type: string
Returns the port direction.

domain

Type: collection
Returns the domain.

full_name

Type: string
Returns the hierarchical name of the UPF supply port.

name

Type: string
Returns the simple name of the UPF supply port.

scope

Type: string
Returns the scope of the UPF supply port, which is the level of logic hierarchy designated as the root of the supply port's domain.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

upf_attributes(3)

upf_supply_set_attributes

Describes the predefined application attributes for upf_supply_set objects.

DESCRIPTION

full_name

Type: string
Returns the hierarchical name of the UPF supply set.

ground

Type: collection
Returns the ground net of the supply set.

name

Type: string
Returns the name of the UPF supply set.

power

Type: collection
Returns the power net of the supply set.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
upf_attributes(3)
```

upf_use_driver_receiver_for_io_voltages

Specifies whether input/output voltages are automatically derived from the driver, receiver, or pad cell.

TYPE

string

DEFAULT

false

DESCRIPTION

You can set this variable to the following values:

- **false** (the default) - Uses the existing user-defined port voltages; does not derive input/output voltages from the driver, receiver, or pad cell. This setting can cause issues related to scaling library groups.
- **true** - Automatically derives input/output voltages from the driver, receiver, or pad cell and overwrites the existing user-defined port voltages.
- **pad** - Automatically derives input/output voltages from the pad cell (is_pad_cell=true) only and overwrites the existing user-defined port voltages.

SEE ALSO

`power_domains_compatibility(3)`

upf_wscript_retain_object_name_scope

Specifies whether the **write_script** command writes out hierarchical scope names.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If this variable is set to **false** (the default), the **write_script** command does not use hierarchical scope names for supply net names in **set_voltage** commands.

If you set this variable to **true**, the **write_script** command writes out hierarchical scope names.

SEE ALSO

`set_voltage(2)`
`write_script(2)`

variation_attributes

Describes the predefined application attributes for variation objects.

DESCRIPTION

full_name

Type: string
Returns the full name of the variation object.

mean

Type: float
Returns the mean of the distribution of the variation.

object_class

Type: string
Returns the class of the object, which is a constant equal to variation. You cannot set this attribute.

skewness

Type: float
Returns the skewness of the distribution of the variation.

std_dev

Type: float
Returns the standard deviation of the distribution of the variation.

summary

Type: string
Returns information about the distribution of the variation, including mean, standard deviation, and skewness.

variance

Type: float
Returns the variance of the distribution of the variation.

SEE ALSO

```
get_attribute(2)
help_attributes(2)
list_attributes(2)
report_attribute(2)
attributes(3)
```

variation_derived_scalar_attribute_mode

Enables get_attribute command to return statistical timing attributes of timing_path and timing_point collections in variation-aware static timing analysis.

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 by the **get_attribute** command are replaced with the mean or quantile values of corresponding variation_arrival, variation_slack, variation_required, or variation_transition attributes. The default is **quantile** in variation-aware analysis. Setting this variable to **none** disables statistical attributes, and **get_attribute** returns the corner values. If variation-aware analysis is turned off, the **get_attribute** command always returns the corner values regardless of the value of this variable.

The **sort_collection** and **filter_collection** are 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 you to create custom timing_path collections that are sorted or filtered by statistical timing attributes.

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

filter_collection(2)
get_attribute(2)
sort_collection(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 you configure the display of the delays appearing in the increment column (labeled Incr).

Set the variable to one of the following values:

- **effective_delay** (the default) - Displays each increment equal to the scalar difference between the arrival values appearing in the path column.
- **delay_variation** - Displays the quantile value (or mean value) of the statistical increment, 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 also obey the **variation_derived_scalar_attribute_mode** variable.

SEE ALSO

report_timing(2)
variation_derived_scalar_attribute_mode(3)

wildcards

Describes supported wildcard characters and ways in which they can be escaped.

DESCRIPTION

The following characters are supported as wildcards:

- Asterisks (*) - Substitute for a string of characters of any length.
- Question marks (?) - Substitute for a single character.

The following commands support wildcard characters:

```
get_cells
get_clocks
get_designs
get_lib_cells
get_lib_pins
get_libs
get_nets
get_pins
get_ports
list_libs
```

In addition to the commands listed, commands that perform an implicit get support wildcard characters.

Escaping Wildcards

Wildcard characters must be escaped using double backslashes (\\) to remove their special regular expression meaning. For more information, see the EXAMPLES section.

Escaping the Escape Character (\\)

This is similar to the escaping wildcard characters; however, the escaping escape character needs one escape character each to escape the escape character. For more information, see the EXAMPLES section.

EXAMPLES

The following examples show how to use wildcard characters.

Using Wildcards

The following example gets all nets in the current design that are prefixed by **in** and followed by any two characters:

```
pt_shell> get_nets in??
{"in11", "in21"}
```

The following example gets all cells in the current design that are prefixed by **U** and followed by a string of characters of any length:

```
pt_shell> get_cells U*
{"U1", "U2", "U3", "U4"}
```

Escaping Wildcards

The following examples show how to use escaping wildcard characters.

The following example gets the **test?1** design in the system.

```
pt_shell> get_designs {test\\?1}
{"test?1"}
```

The same example can be used in a Tcl-based pt_shell using the **list** Tcl command. For example,

```
pt_shell> get_designs [list {test\?1}]
{"test?1"}
```

If neither the curly braces nor the **list** command is used in the Tcl-based pt_shell, the syntax is as follows:

```
pt_shell> get_designs test\\\\?1
{"test?1"}
```

Escaping the Escape Character (\\)

The following examples show how to escape an escape character.

The following example gets the test\1 design in the system.

```
pt_shell> get_designs {test\\\\1}
{"test\1"}
```

The same example as above can be used in the Tcl-based pt_shell by using the **list** Tcl command. For example,

```
pt_shell> get_designs [list {test\\1}]
{"test\1"}
```

If neither curly braces nor the **list** command is used in the Tcl-based pt_shell, the syntax is as follows:

```
pt_shell> get_designs test\\\\\\\\1
{"test\1"}
```

write_script_include_library_constraints

Controls whether constraints set on library objects are written to script output by the **write_script** and **write_sdc** commands.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls whether constraints set on library objects are written to script output by the **write_script** and **write_sdc** commands. When this variable is set to **true** (the default), the tool writes the constraints that meet both of these conditions:

- Constraints that are attached to objects in libraries in use by the current design
- Constraints that were created with the **set_disable_timing** command

SEE ALSO

```
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 this variable is set to **false** (the default), the **write_script** command 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 consist of total capacitance, set by the **set_load** command, and total resistance, set by the **set_resistance** command.

When you set the **write_script_output_lumped_net_annotation** variable to **true**, the **write_script** command outputs lumped network annotations.

If you want intentional **set_resistance** and **set_load** annotations, it is preferred to include them in a separate Tcl script rather than in a Synopsys Design Constraints (SDC) file.

SEE ALSO

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

```
write_script(2)
```