IC Compiler™
Variables and Attributes

Version L-2016.03, March 2016

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.

Third-Party Links

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

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

Copyright Notice for the Command-Line Editing Feature

Copyright Notice for the Line-Editing Library

Contents

DW_lp_op_iso_mode

Specify the datapath gating style for instances of DesignWare low power components that have op_iso_mode parameters.

TYPE

list

DEFAULT

NONE

GROUP

synlib_variables

DESCRIPTION

This variable specifies the datapath gating style inside a low power DesignWare component that has 'op_iso_mode' parameter.

By default, components that have a 'op_iso_mode' parameter have its value set to '0' which allows the 'DW_lp_op_iso_mode' variable to control the datapath gating style.

There are 4 possible values for DW_lp_op_iso_mode: NONE, adaptive, AND, OR. When the value is set to 'NONE', there is no datapath gating happing in the design; When the value is set to 'adaptive', preferred gating style. 'and' or 'or' depending on component When the value is set to 'AND', gating style is 'and'; When the value is set to 'OR', gating style is 'or';

Setting the 'op_iso_mode' parameter to a value other than the default value of '0' (zero) on an instance allows the RTL source code to override the the value set by the 'DW_lp_op_iso_mode' variable.

The op_iso_mode parameter inside a DesignWare component has 5 possible values: 0: apply Design Compiler variable 'DW_lp_op_iso_mode' (default value) 1: 'none' (overrides 'DW_lp_op_iso_mode' setting) 2: 'and' (overrides 'DW_lp_op_iso_mode' setting) 3: 'or' (overrides 'DW_lp_op_iso_mode' setting) 4: preferred gating style. 'and' or 'or' depending on component (overrides 'DW_lp_op_iso_mode' setting)

Use the following command to determine the current value of the variable:

```
prompt> printvar DW_lp_op_iso_mode\f>
```

SEE ALSO

abstraction_enable_power_calculation

Perform power calculations on a design that is to be used as an abstract block.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

The **create_block_abstraction** command processes the block, creates the abstraction information and annotates it on the design in memory. By default, **create_block_abstraction** command invoke the power calculation during it's operation. Subsequent power reporting commands at top level can report power consumption correctly.

Power consumption of the block is calculated during block abstraction. The power data are stored as attributes on the abstract block and they are used by the **report_power** command during the final assembly step of the entire chip. Thus, the **report_power** command is able to report more accurate power consumption for the designs with abstract blocks.

The **abstraction_enable_power_calculation** variable instructs the **create_block_abstraction** command to enable or disable the power calculation.

The **abstraction_enable_power_calculation** variable is on (**true**) by default. If a power license is not available, the power calculation step is disrupted.

Setting this variable as false does not check for power license availability and power calculation step is not invoked.

SEE ALSO

```
create_block_abstraction(2)
report_power(2)
```

abstraction_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

The value you specify for the **abstraction_ignore_percentage** variable defines a minimum threshold for the percentage of the total registers in the transitive fanout of an input port, beyond which the tool is to ignore the port when identifying interface logic. The default is 25.

The tool automatically ignores ports if the ports fan out to a percentage of total registers in the design that is greater than or equal to the value you specify for this variable.

For a discussion about creating block abstractions and related commands, see the man page for the **create_block_abstraction** command.

To see the current value of this variable, type

prompt> **printvar abstraction_ignore_percentage**

SEE ALSO

`create_block_abstraction(2)`

abstraction_max_active_scenarios

Specifies the maximum number of scenarios to be activated simultaneously for operations such as timing update and identifying interface logic while creating ILMs or block abstraction.

TYPE

integer

DEFAULT

0

GROUP

MCMM

DESCRIPTION

This variable controls the runtime or the impact on memory when creating ILMs or block abstraction models by controlling the number of scenarios that are simultaneously activated for operations such as timing update and interface logic identification.

Specify a value between 0 and the maximum number of scenarios (active and inactive) in the design. Note that the value can be more than the value that is specified by using the **set_host_options -max_cores** option.

If the variable is not set or is set to the default 0, the value that is specified by using the **set_host_options -max_cores** option is used.

To reduce the impact on memory:
Use this variable to reduce the potential memory impact when a design has a large number of scenarios. For example, a design has 20 scenarios and the **set_host_options -max_cores** option is set to 16. By default, the **create_ilm** or the **create_block_abstraction** commands activate 16 scenarios and might cause capacity issues. In such cases, set the **abstraction_max_active_scenarios** variable to a smaller value to limit the number of active scenarios while creating ILMs or block abstraction models. Note that the **create_ilm** and **create_block_abstraction** commands process all scenarios unless the **create_ilm -scenarios** command is used. Therefore, the number of

active scenarios that are set by using the **set_active_scenarios** command is not applicable to these commands.

To maximize the runtime benefits:
If the timing of a design is computed before creating ILMs or block abstraction models, set the **abstraction_max_active_scenarios** variable to the number of active scenarios. For example, assume that a design has seven scenarios and the timing of all of them has been updated before creating the ILMs or block abstraction models and the **set_host_options -max_cores** option has been set to 4. By default, the **create_ilm** and **create_block_abstraction** commands activate only the first four scenarios and deactivates the others. Therefore, the computed timing can be reused only for the four scenarios. By setting the **abstraction_max_active_scenarios** variable to 7, the computed timing of all the seven scenarios is reused by the tool, thus giving additional runtime benefit.

SEE ALSO

```
create_block_abstraction(2)
set_host_options(2)
report_host_options(2)
```

access_internal_pins

Controls access of internal pins.

TYPE

Boolean

DEFAULT

true

GROUP

system_variables

DESCRIPTION

The **access_internal_pins** variable controls your access to internal pins. Internal pins are related to the internal design inside a library cell.

When the **access_internal_pins** variable is set to true, you can query internal pins using the **get_pins** command and set attributes or constraints on the internal pins.

SEE ALSO

`get_pins(2)`

alib_library_analysis_path

Specifies a single path, similar to a search path, for reading and writing the alib files that correspond to the target libraries.

TYPE

string

DEFAULT

"./"

DESCRIPTION

This variable specifies the path from which the tool loads alibs during compile. The tool stores alibs to this path when the **alib_analyze_libs** command is issued.

SEE ALSO

attributes

Lists the predefined Synopsys attributes.

DESCRIPTION

Attributes are properties assigned to objects such as nets, cells, and clocks, and describe design features to be considered during optimization.

Attributes are grouped into the following categories:

- cell
- clock
- design
- library cell
- net
- pin
- port
- read-only
- reference

Definitions for these attributes are provided in the subsections that follow.

There are several commands used to set attributes; however, most attributes can be set with the **set_attribute** command. If the attribute definition specifies a set command, use it to set the attribute. Otherwise, use the **set_attribute** command.

Some attributes are informational, or read-only. You cannot set the value of these attributes. Most attribute groups contain read-only attributes; however, a complete list of these attributes is provided in the "Read Only" subsection.

Some attributes are instance-specific, which means they can be applied to specified objects in the design hierarchy. The following attributes are instance-specific:

- disable_timing
- load
- test_assume

Certain attributes are specific to Power Compiler objects. For information about the Power Compiler attributes, see the **power_attributes** man page.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of the attributes, see the man pages of the appropriate set command.

Note that path groups, cell delay, net delay, external delay, point-to-point timing specification, and arrival information are not represented as attributes, and therefore cannot be manipulated with attribute commands.

Cell Attributes

area

Specifies the area of a cell. This attribute does not exist on a hierarchical cell.

The attribute is calculated by the cell's boundary points.

This attribute is read-only and cannot be set by the user.

aspect_ratio

Specifies the *height*:**lidth** ratio of a cell. This attribute does not exist on a hierarchical cell.

This attribute is read-only and cannot be set by the user.

async_set_reset_q

Establishes the value (0 or 1) that should be assigned to the q output of an inferred register if set and reset are both active at the same time. To be used with the **async_set_reset_qn** attribute. Use these attributes if one of the following condition sets apply:

- You have used the **one_hot** or **one_cold** attribute or directive in your HDL description and your logic library is written using pre-V3.0a syntax
- Your logic library does not use a consistent convention for q and qn when set and reset are both active

By default, if set and reset are both active at the same time, Design Compiler uses the convention of the selected logic library, as set with the **target_library** variable. Set this attribute with the **set_attribute** command.

async_set_reset_qn

Establishes the value (0 or 1) that should be assigned to the qn output of an inferred register if set and reset are both active at the same time. To be used with the **async_set_reset_q** attribute. Use these attributes if one of the following condition sets apply:

- You have used the **one_hot** or **one_cold** attribute or directive in your HDL description and your logic library is written using pre-V3.0a syntax
- Your logic library does not use a consistent convention for q and qn when set and reset are both active

If a V3.0a or later syntax logic library is used, then by default, if set and reset are both active at the same time, Design Compiler will use the convention of the selected logic library (**target_library**). Set with **set_attribute**.

If you are unsure whether or not your logic library uses V3.0a syntax, ask your ASIC vendor.

combinational_type_exact

Specifies the replacement gate to use for cells specified in the cell list. Compile attempts to convert combinational gates tagged with **set_compile_type** to the specified replacement combinational gate. Set with **set_combinational_type**.

disable_timing

Disables the timing arcs of a cell. This has the same effect on timing as not having the arc in the library. Set with **set_disable_timing**.

dont_touch

Identifies cells to be excluded from optimization. Values are true (the default) or false. Cells with the **dont_touch** attribute set to true are not modified or replaced during compile. Setting **dont_touch** on a hierarchical cell sets the attribute on all cells below the hierarchical cell. Set with the **set_dont_touchcommand.**

flip_flop_type

Stores the name of the specified flip-flop to be converted from the **target_library**. The **compile** command automatically converts all tagged flip-flops to the specified (or one similar) type. Set with **set_register_type -flip_flop flip_flop_name [cell_list]**.

flip_flop_type_exact

Stores the name of the specified flip-flop to be converted from the **target_library**. The **compile** command automatically converts all tagged flip-flops to the exact flip-flop type. Set with **set_register_type -exact -flip_flop flip_flop_name [cell_list]**.

height

Specifies the height of a cell. This attribute does not exist on a hierarchical cell.

The height is calculated using the cell's cell boundary.

This attribute is read-only and cannot be set by the user.

is_black_box

Sets to true if the cell's reference is not linked to a design or is linked to a design that does not have a functionality.

This attribute is read-only and cannot be set by the user.

is_combinational

Sets to true if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command reports such a cell as not a black-box.

This attribute is read-only and cannot be set by the user.

is_dw_subblock

Sets to true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated.

This attribute is read-only and cannot be set by the user.

Note that DW subblocks that are manually elaborated do not have this attribute.

is_hierarchical

Sets to true if the design is not a leaf design; for example, not from a logic library.

This attribute is read-only and cannot be set by the user.

is_mapped

Sets to true if the cell is not generic logic.

This attribute is read-only and cannot be set by the user.

is_sequential

Sets to true if the cell is sequential. A cell is sequential if it is not combinational.

This attribute is read-only and cannot be set by the user.

is_synlib_module

Sets to true if the object (a cell, a reference, or a design) refers to an unmapped module reference, or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator.

This attribute is read-only and cannot be set by the user.

Note that synlib modules that are manually elaborated do not have this attribute.

is_synlib_operator

Sets to true if the object (a cell or a reference) is a synthetic library operator reference.

This attribute is read-only and cannot be set by the user.

is_test_circuitry

Sets by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry.

This attribute is read-only and cannot be set by the user.

is_unmapped

Sets to true if the cell is generic logic.

This attribute is read-only and cannot be set by the user.

latch_type_exact

Stores the name of the specified latch to be converted from the **target_library**. The **compile** command automatically converts all tagged latches to the exact latch type. Set with **set_register_type -latch latch_name [cell_list]**.

macro_area_percentage

Specifies the percentage of the total macro area of a soft macro's physical area.

This attribute is read-only and cannot be set by the user.

map_only

Specifies that **compile** attempts to map the object exactly in the target library, and exclude the object from logic-level optimization (flattening and structuring) when set to true. The default is false. Set with **set_map_only**.

max_fall_delay

Specifies a floating-point value that establishes the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_metal_layer

Specifies the reserved maximum metal layer name of a soft macro or a black box.

This attribute is read-only and cannot be set by the user.

max_rise_delay

Specifies a floating-point value that establishes the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow

Specifies 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 **set_max_time_borrow**.

min_fall_delay

Specifies a floating-point value that establishes the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_metal_layer

Specifies the reserved minimum metal layer name of a soft macro or a black box.

This attribute is read-only and cannot be set by the user.

min_rise_delay

Specifies a floating-point value that establishes the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

number_of_black_box

Specifies the count of black boxes in a hierarchical cell or a soft macro. Whether a cell is a black box can be determined by the attribute **is_black_box**.

This attribute is read-only and cannot be set by the user.

number_of_io_cell

Specifies the count of I/O cells in a hierarchical cell or a soft macro.

Cells are counted in when the **mask_layout_type** is either **io_pad** or **corner_pad**.

This attribute is read-only and cannot be set by the user.

number_of_macro

Specifies the count of macros in a hierarchical cell or a soft macro.

Cells are counted in when the **mask_layout_type** is **macro**.

This attribute is read-only and cannot be set by the user.

number_of_pinshape

Specifies the number of pin shapes of a soft macro.

This attribute is read-only and cannot be set by the user.

number_of_standard_cell

Specifies the count of standard cells in a hierarchical cell or a soft macro.

Cells are counted in when the **mask_layout_type** is **std**.

This attribute is read-only and cannot be set by the user.

physical_area

Specifies the physical area of a hierarchical cell or a soft macro.

The physical area of a hierarchical cell is the sum of its direct children's **physical_area** or **area**. If a direct child is a hierarchical cell, then **physical_area** is used. If a child is a standard cell or macro, then **area** is used, otherwise that child is skipped.

The physical area of a soft macro is the sum of its children's **physical_area** or **area**. The children are iterated from the subdesign file. If a child is a soft macro, then **physical_area** is used. if a child is a standard cell or hard macro, **area** is used, otherwise that child is skipped.

This attribute is read-only and cannot be set by the user.

physical_area_percentage_in_top_design

Specifies the percentage of **physical_area** of a soft macro in the top design.

This attribute is read-only and cannot be set by the user.

ref_name

Specifies the reference name of a cell.

This attribute is read-only and cannot be set by the user.

full_name

Specifies the hierarchical name of cell, pin or net.

This attribute is read-only and cannot be set by the user.

scan

Specifies that the cell is always replaced by an equivalent scan cell during **insert_dft** when set to true. When set to false, the cell is not replaced. Set with **set_scan**.

scan_chain

Includes the specified cells of the referenced design in the scan-chain whose index is the value of this attribute. Set with **set_scan_chain**.

scan_element

Determines if sequential cells in the specified designs are replaced by equivalent scan cells during **insert_scan**. When true, the default, **insert_scan** replaces cell_design_ref_list with equivalent scan cells. The scan cells are not replaced when set to false. Set with **set_scan_element**.

scan_latch_transparent

Makes specified cells transparent during ATPG when set to true. For hierarchical cells, the effects apply hierarchically to level-sensitive leave cells. Set with **set_scan_transparent**. Remove with **remove_attribute**.

test_isolate

Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.

Note that setting this attribute suppresses the warning messages associated with the isolated objects.

test_routing_position

Specifies the preferred routing order of the scan-test signals of the identified cells. Set with **set_test_routing_order**.

ungroup

Removes a level of hierarchy by exploding the contents of the specified cell in the current design. If specified on a reference object, cells using that reference are ungrouped during **compile**. Set with **set_ungroup**.

register_list

Specifies the single-bit cells that are remapped to the multibit cell. This attribute can be set when the compile_ultra or the create_regsiter_bank commands remap single-bit registers to multibit registers.

Clock Attributes

clock_fall_transition

Sets the falling transition value on the specified clock list. The clock_fall_transition overrides the calculated transition times on clock pins of registers and associated nets. Set using **set_clock_transition**.

clock_rise_transition

Sets the rising transition value on the specified clock list. The clock_rise_transition overrides the calculated transition times on clock pins of registers and associated nets. Set using **set_clock_transition**.

dont_touch_network

When a design is optimized, **compile** assigns **dont_touch** attributes to all cells and nets in the transitive fanout of **dont_touch_network** ports. The **dont_touch** assignment stops at the boundary of storage elements. An element is recognized as storage only if it has setup or hold constraints. Set with **set_dont_touch_network.**

fix_hold

Specifies that **compile** should attempt to fix hold violations for timing endpoints related to this clock. Set with **set_fix_hold**.

hold_uncertainty

Specifies a negative uncertainty from the edges of the ideal clock waveform. Set with **set_clock_uncertainty -hold**.

max_fall_delay

A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_rise_delay

A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow

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 **set_max_time_borrow**.

min_fall_delay

A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay

A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

period

Assigns a value to the clock period. The clock period (or cycle time) 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. Set with **create_clock -period_value**.

propagated_clock

Specifies that the clock edge times be delayed by propagating the values through the clock network. If this attribute is not present, ideal clocking is assumed. Set with **set_propagated_clock**.

setup_uncertainty

Specifies a positive uncertainty from the edges of the ideal clock waveform. Set with **set_clock_uncertainty -setup**.

Design Attributes

async_set_reset_q

Establishes the value (0 or 1) that should be assigned to the q output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_qn**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your logic library is written using pre-V3.0a syntax; *or* if your logic library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax logic library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected logic library (**target_library**). Set with **set_attribute**.

Note: If you are unsure whether or not your logic library uses V3.0a syntax, ask your ASIC vendor.

async_set_reset_qn

Establishes the value (0 or 1) that should be assigned to the qn output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_q**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your logic library is written using pre-V3.0a syntax; *or* if your logic library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax logic library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected logic library (**target_library**). Set with **set_attribute**.

If you are unsure whether or not your logic library uses V3.0a syntax, ask your ASIC vendor.

balance_registers

Determines whether the registers in a design are retimed during **compile**. When **true** (the default value), **compile** invokes the **balance_registers** command, which moves registers to minimize the maximum register-to-register delay. Set this attribute to **false**, or remove it, to disable this behavior.

Set with **set_balance_registers**.

If your design contains generic logic, you should ensure that all components are mapped to cells from the library before setting the **balance_registers** attribute.

boundary_optimization

Enables **compile** to optimize across hierarchical boundaries. Hierarchy is ignored during optimization for designs with this attribute set to **true**. Set with **set_boundary_optimization .**

default_flip_flop_type

Specifies the default flip-flop type for the current design. During the mapping process, **compile** tries to convert all unmapped flip-flops to this type. If **compile** is unable to use this flip-flop, it maps these cells into the smallest flip-flop possible. Set with **set_register_type -flip_flop flip_flop_name**.

default_flip_flop_type_exact

During the mapping process, **compile** converts unmapped flip-flops to the exact flip-flop type specified here. Set with **set_register_type -exact -flip_flop flip_flop_name**

default_latch_type_exact

Specifies the exact default latch type for the **current_design**. During the mapping process, **compile** converts unmapped latches to the exact latch type specified here. Set with **set_register_type -exact -latch latch_name**.

design_type

Indicates the current state of the design and has the value **fsm** (finite state machine), **pla** (programmable logic array), **equation** (Boolean logic), or **netlist** (gates). This attribute is "read-only" and cannot be set by the user.

dont_touch

Identifies designs that are to be excluded from optimization. Values are **true** (the default) or **false**. Designs with the **dont_touch** attribute set to **true** are not modified or replaced during **compile**. Setting **dont_touch** on a design has an effect only when the design is instantiated within another design as a level of hierarchy; setting **dont_touch** on the top-level design has no effect. Set with the **set_dont_touch** command.

flatten

When set to **true**, determines that a design is to be flattened during **compile**. By default, a design is not flattened. Set with the **set_flatten** command.

flatten_effort

Defines the level of CPU effort that **compile** uses to flatten a design. Allowed values are **low** (the default), **medium**, or **high**. Set with the **set_flatten** command.

flatten_minimize

Defines the minimization strategy used for logic equations. Allowed values are **single_output**, **multiple_output**, or **none**. Set with the **set_flatten** command.

flatten_phase

When **true**, allows logic flattening to invert the phase of outputs during **compile**. By default, logic flattening does not invert the phase of outputs. Used only if the **flatten** attribute is set. Set with **set_flatten.**

implementation

The implementation for each specified instance of the specified component_type. Specifying the **-default** option removes this attribute from all instances of the component type in the current design. Set with **set_jtag_implementation**.

is_combinational

true if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or nonthree-state and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute is read-only; you cannot set it.

is_dw_subblock

true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute is "read-only" and cannot be set by the user.

NOTE: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical

true if any of the cells of a design are not leaf cells (for example, not from a logic library). This attribute is read-only and cannot be set by the user.

is_mapped

true if all the non-hierarchical cells of a design are mapped to cells in a logic library. This attribute is "read-only" and cannot be set by the user.

is_sequential

true if any cells of a design or designs in its hierarchy are sequential. A cell is sequential if it is not combinational (if any of its outputs depend on previous inputs). This attribute is read-only and cannot be set by the user.

is_synlib_module

true if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is "read-only" and cannot be set by the user.

NOTE: synlib modules that are manually elaborated will not have this attribute.

is_unmapped

true if any of the cells are not linked to a design or mapped to a logic library. This attribute is read-only and cannot be set by the user.

local_link_library

A string that contains a list of design files and libraries to be added to the beginning of the **link_library** whenever a **link** operation is performed. Set with **set_local_link_library**.

map_only

When set to **true**, **compile** will attempt to map the object exactly in the target library, and will exclude the object from logic-level optimization (flattening and structuring). The default is **false**. Set with **set_map_only**.

max_area

A floating point number that represents the target area of the design. **compile** uses it to calculate the area cost of the design. The units must be consistent with the units used from the logic library during optimization. Set with **set_max_area**.

max_capacitance

A floating point number that sets the maximum capacitance value for input, output, or bidirectional ports, or designs. The units must be consistent with those of the logic library used during optimization. Set with **set_max_capacitance**.

max_dynamic_power

A floating point number that specifies the maximum target dynamic power for the **current_design**. The units must be consistent with those of the logic library. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_dynamic_power**.

max_leakage_power

A floating point number that specifies the maximum target leakage power for the **current_design**. The units must be consistent with those of the logic library. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_leakage_power**.

max_total_power

A floating point number that specifies the maximum target total power for the **current_design**. Total power is defined as the sum of dynamic and leakage power. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_total_power**.

minimize_tree_delay

When **true** (the default value), **compile** restructures expression trees in the **current_design** or in a list of specified designs, to minimize tree delay. The value of this attribute overrides the value of **hlo_minimize_tree_delay**. Set this attribute to **false** for any designs that you do not wish to be restructured. Set with **set_minimize_tree_delay**.

model_map_effort

Specifies the relative amount of CPU time to be used by **compile** during modeling, typically for synthetic library implementations. Values are **low**, **medium**, and **high**, or 1, 2, and 3. If **model_map_effort** is not set, the value of **synlib_model_map_effort** is used. Set with the **set_model_map_effort** command.

model_scale

A floating point number that sets the model scale factor for the **current_design**. Set with **set_model_scale**.

optimize_registers

When **true** (the default value), **compile** automatically invokes the Behavioral Compiler **optimize_registers** command to retime the design during optimization. Setting the attribute to **false** disables this behavior. Your design cannot contain generic logic at the instant **optimize_registers** is invoked during **compile**.

Set with **set_optimize_registers**.

part

A string value that specifies the Xilinx part type for a design. For valid part types, refer to the Xilinx *XC4000 Databook*. Set with **set_attribute**.

port_is_pad

Indicates specified ports are to have I/O pads attached. The I/O pads are added during insert_pads and automatically added during compile. Set using **set_port_is_pad**.

resource_allocation

Indicates the type of resource allocation to be used by **compile** for the **current_design**. Allowed values are **none**, indicating no resource sharing; **area_only**, indicating resource sharing with tree balancing without considering timing constraints; **area_no_tree_balancing**, indicating resource sharing without tree balancing and without considering timing constraints; and **constraint_driven** (the default), indicating resource sharing so that timing constraints are met or not worsened. The value of this attribute overrides the value of the variable **hlo_resource_allocation** for the **current_design**. Set with **set_resource_allocation**.

resource_implementation

Indicates the type of resource implementation to be used by **compile** for the **current_design**. Allowed values are **area_only**, indicating resource implementation without considering timing constraints; **constraint_driven**, indicating resource implementation so that timing constraints are met or not worsened; and **use_fastest**, indicating resource implementation using the fastest implementation initially, unless all timing constraints are met. If the fastest implementation has been selected initially later steps of the compile command will select components with smaller area later in uncritical parts of the design. The value of this attribute overrides the value of the variable **hlo_resource_implementation** for the **current_design**. Set with **set_resource_implementation**.

scan_element

Determines if sequential cells in the specified designs are replaced by equivalent scan cells or designs during **insert_scan**. Default is set to **true**. When set to **false**, sequential cells are not replaced by equivalent scan cells. Set with **set_scan_element**.

scan_latch_transparent

When set to **true**, makes specified designs transparent during ATPG. For hierarchical cells, the effects apply hierarchically to level-sensitive leaf cells. The **set_scan_transparent** command sets the attribute; the **remove_attribute** command removes it.

share_cse

When **true**, the value of the environment variable **hlo_share_common_subexpressions** is used. The value of this attribute determines whether common subexpressions are shared during compile, to reduce the cost of the design. Setting the attribute to **false** overrides the **hlo_share_common_subexpressions**. Set with **set_share_cse**.

structure

Determines if a design is to be structured during **compile**. If **true**, adds logic structure to a design by adding intermediate variables that are factored out of the design's equations. Set with **set_structure .**

structure_boolean

Enables the use of Boolean (non-algebraic) techniques during the structuring phase of optimization. This attribute is ignored if the **structure** attribute is **false** Set with **set_structure .**

structure_timing

Enables timing constraints to be considered during the structuring phase of optimization. This attribute is ignored if the **structure** attribute is **false**. Set with **set_structure .**

ungroup

Removes a level of hierarchy from the current design by exploding the contents of the specified cell in the current design. Set with **set_ungroup .**

wired_logic_disable

When **true**, disables creation of wired OR logic during **compile**. The default is **false**; if this attribute is not set, wired OR logic will be created if appropriate. Set with **set_wired_logic_disable**.

wire_load_model_mode

Determines which wire load model to use 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 are **top**, which indicates to use the wire load model at the top hierarchical level; **enclosed**, which indicates to use the wire load model on the smallest design that encloses a net completely; and **segmented**, which indicates to break 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 a value is not specified for this attribute, **compile** searches for a default in the first library in the link path. If none is found, **top** is the default. Set with **set_wire_load**.

xnfout_use_blknames

When **true**, the Synopsys XNF writer writes BLKNM XNF parameters into the XNF netlist for your design when **write -f xnf** is invoked. The default is **false**. The BLKNM XNF parameters convey to the Xilinx place and route tools information, previously placed on the **db_design** by **replace_fpga**, that indicates which groupings of function generators are to be packed into CLB cells. Set with **set_attribute**.

Library Cell Attributes

dont_touch

Identifies library cells to be excluded from optimization. Values are **true** (the default) or **false**. Library cells with the **dont_touch** attribute set to **true** are not modified or replaced during **compile**. Setting **dont_touch** on a hierarchical cell sets the attribute on all cells below it. Set with **set_dont_touch .**

dont_use

Disables the specified library cells so that they are not added to a design during **compile**. Set with **set_dont_use .**

formula

The attribute of the priority parameter for implementations in synthetic libraries. The formula should evaluate to an integer between 0 and 10. Set with **set_impl_priority**.

implementation

Specifies the implementation for the synthetic library cell instances to use. When compile is run, the implementation you specified is used if you set this attribute. The cells instances must be defined in the synthetic library for this attribute to work. Set with **set_implementation**.

no_sequential_degenerates

When **true**, disables mapping to versions of this latch or flip flop that have some input pins connected to 0 or to 1. Set with **set_attribute**. This attribute may also be set on the library itself, and that value will apply as the default for all registers in the library which don't have the attribute set individually.

preferred

Specifies the preferred library gate to use during technology translation when there are other gates with the same function in the target library. Set with the **set_prefer** command.

scan

When **true**, specifies that the instances of the library cell are always replaced by equivalent scan cells during **insert_dft**. When false, instances are not replaced. Set with **set_scan.**

scan_group

A user-defined string variable that allows you to specify to DFT Compiler a preferred scan equivalent for a non-scan storage element, when a library contains multiple scan equivalents. Typical values are **low**, **medium**, and **high**, for low, medium, and high drive strengths. However, you can define any string variable, and it need not describe drive strength. The default behavior is for DFT Compiler to attempt to choose a scan element that best matches the electrical characteristics of the nonscan element; for a more detailed explanation, refer to the *DFT Compiler Scan Synthesis User Guide*. The matching of electrical characteristics works well with the standard CMOS delay model, but is not accurate with other delay models; **scan_group** provides a means for you to specify an appropriate scan equivalent. Normally, **scan_group** would be set by the ASIC vendor or library developer, but can also be set by you. Consult your ASIC vendor before attempting to set **scan_group** with **set_attribute**. For more information about **scan_group**, refer to the *DFT Compiler Scan Synthesis User Guide*.

set_id

Allows for the value for the implementations in synthetic libraries. Set with **set_impl_priorities**.

scan_element

Determines if specified designs are scan replaced by **insert_scan**. Set using **set_scan_element**.

scan_latch_transparent

When **true**, makes the specified library cells transparent in ATPG. For hierarchical cells, the effects apply hierarchically to level-sensitive leaf cells. The **set_scan_transparent** command sets the attribute; the **remove_attribute** command removes it.

sequential_bridging

When **true**, enables **Design Compiler** to take a multiplexed flip-flop and bridge (that is, connect) the output to the input to get a desired functionality. The default is **false**, so this attribute must be set in order to enable the functionality. Bridging is required for mapping in cases where there is no flip-flop with internal feedback in the target library but one is desired in the HDL. Set with **set_attribute**. This attribute may also be set on the library itself, and that value will apply as the default for all registers in the library which don't have the attribute set individually.

NOTE: Setting this attribute to **true** can result in an increase in run times and memory consumption for Design Compiler. The increased run times depend on the number of flip-flops in the target library or libraries for which this attribute has been set.

Pin Attributes

actual_max_net_capacitance

actual_min_net_capacitance

A floating point number that specified the total calculated capacitance of the net that is connected to the given pin. The attributes are defined only for pins of leaf cell. The value of these attributes is calculated upon request. These are "read-only" attributes and they cannot be set by the user.

disable_timing

Disables timing arcs. This has the same effect on timing as not having the arc in the library. Set with **set_disable_timing.**

max_slack

A floating point value representing the worst slack of **max_rise_slack** and **max_fall_slack**.

max_fall_slack

A floating point value representing the worst slack at a pin for falling maximum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

max_rise_slack

A floating point value representing the worst slack at a pin for rising maximum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

min_slack

A floating point value representing the worst slack of **min_rise_slack** and **min_fall_slack**.

min_fall_slack

A floating point value representing the worst slack at a pin for falling minimum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated..

min_rise_slack

A floating point value representing the worst slack at a pin for rising minimum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

max_fall_delay

A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_rise_delay

A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

min_fall_delay

A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay

A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

hold_uncertainty

Specifies a negative uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this pin. Set with **set_clock_uncertainty -hold**.

observe_pin

Specifies the (internal) observe pin name of an LSI Logic scan macrocell (LSI CTV only). This attribute is used by the **write_test** command. Set with **set_attribute .**

pin_direction

Specifies the direction of a pin. Allowed values are **in**, **out**, **inout**, or **unknown**. This attribute is **read-only** and cannot be set by the user.

pin_properties

Lists valid EDIF property values to be attached to different versions of the output pin. The EDIF property values correspond to different output emitter-follower resistance values on the output pin. For details about the use of this attribute, refer to the *Library Compiler Reference Manual*, Chapter 6, "Defining Cells." Set with **set_attribute**.

setup_uncertainty

Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this pin. Set with **set_clock_uncertainty -setup**.

set_pin

Specifies the (internal) set pin name of an LSI Logic scan macrocell (LSI CTV only). This attribute is used by the **write_test** command. Set with **set_attribute .**

signal_type

Used to indicate that a pin or port is of a special type, such as a **clocked_on_also** port in a master/slave clocking scheme, or a **test_scan_in** pin for scan-test circuitry. Set with **set_signal_type .**

static_probability

A floating point number that specifies the percentage of time that the signal is in the logic 1 state; this information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

test_assume

A string that represents a constant logic value to be assumed for specified pins throughout test design rule checking by **check_test**. "1", "one", or "ONE" specifies a constant value of logic one; "0", "zero", or "ZERO" specifies a constant value of logic zero. Use **report_test -assertions** for a report on objects that have the **test_assume** attribute set. Set with **set_test_assume .**

test_initial

A string that represents an initial logic value to be assumed for specified pins at the start of test design rule checking and fault simulation by **check_test**. "1", "one", or "ONE" specifies an initial value of logic one; "0", "zero", or "ZERO" specifies an initial value of logic zero. Use **report_test -assertions** for a report on objects that have the **test_initial** attribute set. Set with **set_test_initial**.

test_isolate

Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.

Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

test_routing_position

Specifies the preferred routing order of the scan-test signals of the identified cells. Set with **set_test_routing_order .**

toggle_rate

A positive floating point number that specifies the toggle rate; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**; if this attribute is not set, **report_power** will use the default value of 2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

true_delay_case_analysis

Specifies a value to set all or part of an input vector for **report_timing -true** and **report_timing -justify**. Allowed values are **0**, **1**, **r** (rise, X to 1), and **f** (fall, X to 0). Set with the **set_true_delay_case_analysis** command.

Port Attributes

actual_max_net_capacitance

actual_min_net_capacitance

A floating point number that specified the total calculated capacitance of the net connected to the given port. The value of these attributes is calculated upon request. These are "read-only" attributes and they cannot be set by the user.

connection_class

A string that specifies the connection class label to be attached to a port or to a list of ports. **compile**, **insert_pads**, and **insert_dft** will connect only those loads and drivers that have the same connection class label. The labels must match those in the library of components for the design, and must be separated by a space. The labels **universal** and **default** are reserved; **universal** indicates that the port can connect with any other load or driver, and

default is assigned to any ports that do not have a connection class already assigned. Set with **set_connection_class**.

dont_touch_network

When a design is optimized, **compile** assigns **dont_touch** attributes to all cells and nets in the transitive fanout of **dont_touch_network** clock objects. The **dont_touch** assignment stops at the boundary of storage elements. An element is recognized as storage only if it has setup or hold constraints. Set with **set_dont_touch_network .**

driven_by_dont_care

Specifies that input port are driven by dont_care. Compile uses this information to create smaller designs. After optimization, the port connected to dont_care does not drive anything inside the optimized design. Set with **set_logic_dc**.

driven_by_logic_one

Specifies that input ports are driven by logic one. **compile** uses this information to create smaller designs. After optimization, a port connected to logic one usually does not drive anything inside the optimized design. Set with **set_logic_one .**

driven_by_logic_zero

Specifies that input ports are driven by logic zero. **compile** uses this information to create smaller designs. After optimization, a port connected to logic zero usually does not drive anything inside the optimized design. Set with **set_logic_zero .**

driving_cell_dont_scale

When **true**, indicates not to scale the transition time on the port using the driving cell. Otherwise the transition time will be scaled by operating condition factors. Set with **set_driving_cell**.

driving_cell_fall

A string that names a library cell from which to copy fall drive capability to be used in fall transition calculation for the port. Set with **set_driving_cell**.

driving_cell_from_pin_fall

A string that names the driving_cell_fall input pin to be used to find timing arc fall drive capability. Set with **set_driving_cell**.

driving_cell_from_pin_rise

A string that names the driving_cell_rise input pin to be used to find timing arc rise drive capability. Set with **set_driving_cell**.

driving_cell_library_fall

A string that names the library in which to find the **driving_cell_fall**. Set with **set_driving_cell**.

driving_cell_library_rise

A string that names the library in which to find the **driving_cell_rise**. Set with **set_driving_cell**.

driving_cell_multiply_by

A floating point value by which to multiply the transition time of the port marked with this attribute. Set with **set_driving_cell**.

driving_cell_pin_fall

A string that names the driving_cell_fall output pin to be used to find timing arc fall drive capability. Set with **set_driving_cell**.

driving_cell_pin_rise

A string that names the driving_cell_rise output pin to be used to find timing arc rise drive capability. Set with **set_driving_cell**.

driving_cell_rise

A string that names a library cell from which to copy rise drive capability to be used in rise transition calculation for the port. Set with **set_driving_cell**.

fall_drive

Specifies the drive value of high to low transition on input or inout ports. Set with **set_drive.**

fanout_load

Specifies the fanout load on output ports. Set with **set_fanout_load .**

load

Specifies the load value on ports. The total load on a net is the sum of all the loads on pins, ports, and wires associated with that net. Set with **set_load.**

max_capacitance

A floating point number that sets the maximum capacitance value for input, output, or bidirectional ports, and/or designs. The units must be consistent with those of the logic library used during optimization. Set with **set_max_capacitance**.

max_fall_delay

A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_fanout

Specifies the maximum fanout load for the net connected to this port. **compile** ensures that the fanout load on this net is less than the specified value. Set with **set_max_fanout .**

max_rise_delay

A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_slack

A floating point value representing the worst slack of **max_rise_slack** and **max_fall_slack**.

max_fall_slack

A floating point value representing the worst slack at a pin for falling maximum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

max_rise_slack

A floating point value representing the worst slack at a pin for rising maximum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

min_slack

A floating point value representing the worst slack of **min_rise_slack** and **min_fall_slack**.

min_fall_slack

A floating point value representing the worst slack at a pin for falling minimum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

min_rise_slack

A floating point value representing the worst slack at a pin for rising minimum path delays. This attribute is valid for any pin that appears in a constrained path after timing has been updated.

max_time_borrow

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 **set_max_time_borrow**.

max_transition

Specifies the maximum transition time for the net connected to this port. **compile** ensures that value. Set with **set_max_transition .**

min_capacitance

A floating point number that sets the minimum capacitance value for input and/or bidirectional ports. The units must be consistent with those of the logic library used during optimization. Set with **set_min_capacitance**.

min_fall_delay

A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay

A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

setup_uncertainty

Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_uncertainty -setup**.

hold_uncertainty

Specifies a negative uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_uncertainty -hold**.

model_drive

A non-negative floating point number that specifies the estimated drive value on ports in terms of standard drives of the current logic library. Set with **set_model_drive**.

model_load

A non-negative floating point number that specifies the estimated load value on ports in terms of standard loads of the current logic library. Set with **set_model_load**.

op_used_in_normal_op

Specifies that a scan-out port is also used in normal operation (system mode). This attribute is used by the **insert_dft** command. Set with **set_attribute**.

Read-Only Attributes

design_type

Indicates the current state of the design and has the value **fsm** (finite state machine), **pla** (programmable logic array), **equation** (Boolean logic), or **netlist** (gates). This attribute cannot be set by the user.

is_black_box

true if the reference is not yet linked to a design or is linked to a design that doesn't have a functionality. This attribute cannot be set by the user.

is_combinational

true if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute is read-only and cannot be set by the user.

is_dw_subblock

true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute is "read-only" and cannot be set by the user.

is_hierarchical

true if any of the cells of a design are not leaf cells (for example, not from a logic library). This attribute cannot be set by the user.

is_mapped

true if all the non-hierarchical cells of a design are mapped to cells in a logic library. This attribute cannot be set by the user.

is_sequential

true if any cells of a design or designs in its hierarchy are sequential. A cell is sequential if it is not combinational. This attribute cannot be set by the user.

is_synlib_module

true if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is "read-only" and cannot be set by the user.

NOTE: synlib modules that are manually elaborated will not have this attribute.

is_synlib_operator

true if the object (a cell or a reference) is a synthetic library operator reference. This attribute is "read-only" and cannot be set by the user.

is_test_circuitry

Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry. This attribute cannot be set by the user.

is_unmapped

true if any of the cells are not linked to a design or mapped to a logic library. This attribute cannot be set by the user.

direction

Returns the direction of a pin or port. The value can be **in**, **out**, **inout**, or **unknown**. For backward compatibility, the attribute also supports an integer value, **1** for **in**, **2** for **out**, or **3** for **inout**. This attribute cannot be set by the user.

pin_direction

Returns the direction of a pin. The value can be **in**, **out**, **inout**, or **unknown**. This attribute cannot be set by the user. You can also use the **direction** attribute to get the direction of the pin.

port_direction

Returns the direction of a port. The value can be **in**, **out**, **inout**, or **unknown**. This attribute cannot be set by the user. You can also use the **direction** attribute to get the direction of the port.

ref_name

The reference name of a cell. This attribute cannot be set by the user.

Reference Attributes

dont_touch

Specifies that designs linked to a reference with this attribute are excluded from optimization. Values are **true** (the default) or **false**. Designs linked to a reference with the **dont_touch** attribute set to **true** are not modified or replaced during **compile**. Set with **set_dont_touch .**

is_black_box

true if the reference is not yet linked to a design or is linked to a design that doesn't have a functionality. This attribute is read-only and cannot be set by the user.

is_combinational

true if all the cells of the referenced design are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute is read-only and cannot be set by the user.

is_dw_subblock

true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute is read-only and cannot be set by the user.

NOTE: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical

true if the referenced design is not a leaf cell (for example, not in a logic library). This attribute is read-only and cannot be set by the user.

is_mapped

true if the reference is linked to a design, and all the non-hierarchical cells of the referenced design are mapped to cells in a logic library. This attribute is read-only and cannot be set by the user.

is_sequential

true if all the cells of the referenced design are sequential. A cell is sequential if it is not combinational (if any of its outputs depend on previous inputs). This attribute is read-only and cannot be set by the user.

is_synlib_module

true if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is read-only and cannot be set by the user.

NOTE: synlib modules that are manually elaborated will not have this attribute.

is_synlib_operator

true if the object (a cell or a reference) is a synthetic library operator reference. This attribute is read-only and cannot be set by the user.

is_unmapped

true if any of the non-hierarchical cells of the referenced design are not mapped to cells in a logic library, or if the reference is not yet linked to a design. This attribute is read-only and cannot be set by the user.

scan

When **true**, specifies that cells of the referenced design are always replaced by equivalent scan cells during **insert_dft**. When false, cells of the design are not replaced. Set with **set_scan .**

scan_chain

Includes the specified cells of the referenced design in the scan-chain whose index is the value of this attribute. Set with the **set_scan_chain** command.

scan_element

Determines if specified designs are scan replaced by **insert_scan**. Set using **set_scan_element**.

scan_latch_transparent

When **true**, makes the specified references transparent in ATPG. For hierarchical cells, the effects apply hierarchically to level-sensitive leaf cells. The specified library cell cannot be overwritten. Set with **set_scan_transparent**; remove with **remove_attribute**.

ungroup

Specifies that all designs linked to a reference with this attribute are ungrouped (levels of hierarchy represented by these design cells are removed) during **compile**. Set with **set_ungroup .**

utilization

Specifies the utilization of a soft macro.

We calculate the utilization using **physical_area**:**area** ratio of a soft macro.

This attribute is "read-only" and cannot be set by the user.

width

Specifies the width of a cell. This attribute doesn't exist on a hierarchical cell.

We use the cell's cell boundary to calculate its width.

This attribute is "read-only" and cannot be set by the user.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
set_attribute(2)
power_attributes(3)
```

auto_insert_level_shifters

Controls automatic level shifter insertion.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

By default, this variable is **true** and enables the level-shifter insertion engine. With this variable set to **true**, commands such as **compile** and **insert_dft** are able to automatically insert level shifters where needed.

Setting this variable to **false** disables the level-shifter insertion engine, and automatic level shifter insertion does not happen during **compile** or **insert_dft** regardless of any user constraint.

Note that level-shifter insertion performed by the command **insert_mv_cells** is not affected by this variable.

SEE ALSO

`insert_mv_cells(2)`

auto_insert_level_shifters_on_clocks

Directs automatic level-shifter insertion to insert level shifters on specified clocks.

TYPE

string

DEFAULT

all

GROUP

none

DESCRIPTION

This variable can be set to "all" or a list of clock names (delimited by spaces or commas). When set to "all", automatic level-shifter insertion inserts level shifters on all clock nets that need level shifters.

When this variable is set to a list of clock names, automatic level-shifter insertion inserts level shifters on the net of these clocks, if needed.

By default, automatic level-shifter insertion does not insert level shifters on the nets driven by the clocks.

Note that if a net is ideal, automatic level-shifter insertion won't insert any level shifter on it, unless the variable **mv_insert_level_shifters_on_ideal_nets** is set to "all".

SEE ALSO

```
auto_insert_level_shifters(3)
create_clock(2)
mv_insert_level_shifters_on_ideal_nets(3)
set_ideal_network(2)
```

auto_link_disable

Specifies whether to disable the code to perform an auto_link during any command.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

When this variable is set to true, the code to perform an auto_link during any command is disabled, resulting in faster command processing. This increase in speed is important in back-annotation commands such as **set_load**, **set_resistance**, and **set_annotated_delay**, where numerous commands are executed in sequence. Disabling the auto_link code can significantly improve the speed with which commands are executed.

Once the sequence of time-critical commands is completed, reset the variable value to false to revert the tool back to its normal mode of operation.

To determine the current value of this variable use the **printvar auto_link_disable** command.

SEE ALSO

```
set_annotated_delay(2)
set_load(2)
set_resistance(2)
```

auto_link_options

Specifies the **link** command options to be used when **link** is invoked automatically.

TYPE

string

DEFAULT

-all

GROUP

system_variables

DESCRIPTION

This variable specifies the **link** command options to be used when **link** is invoked automatically. The default value is -all. To find the available options, refer to the **link** command man page.

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

SEE ALSO

`link(2)`

auto_ungroup_preserve_constraints

Controls whether the timing constraints on the hierarchy are preserved when the hierarchy is ungrouped during the process of optimization.

TYPE

Boolean

DEFAULT

true

GROUP

timing

DESCRIPTION

This variable enables the ungrouping of hierarchies with timing constraints and preserves the timing constraints when the hierarchies are ungrouped during the process of optimization. By default, the **auto_ungroup_preserve_constraints** variable is set to true.

The constraints are preserved when executing the following commands:

```
prompt> compile -ungroup_all

prompt> set_ungroup
prompt> compile

prompt> compile -auto_ungroup area | delay
```

The constraints on DesignWare library hierarchical cells are also lost when the cells are ungrouped during optimization.

Set the **auto_ungroup_preserve_constraints** variable to false before compiling to avoid the ungrouping of hierarchies with timing constraints.

Use the **printvar auto_ungroup_preserve_constraints** command to determine the current value of this variable.

SEE ALSO

```
list(2)
set_ungroup(2)
```

auto_wire_load_selection

Controls automatic selection of wire load model.

TYPE

string

DEFAULT

true

GROUP

`compile_variables`

DESCRIPTION

When area_locked, the automatic wire load selection uses the initial area to do the first selection of the wire load model and then adjusts the wire load model down if the area drops. When area_reselect, the automatic wire load selection during reporting and at various points in compile updates the wire load model to the current area of the design. When false, the automatic wire load selection is off. For backwards compatibility, we also support the value true. True is the same as area_locked.

The automatic selection of the wire load model is used to estimate net capacitances and resistances from the net fanout. 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 will be chosen based on the area of the design containing the net either partially (for **segmented**) or fully (for **enclosed**). If the wire load mode is **top**, the wire load model will be chosen based on the area of the top level design for all nets in the design hierarchy.

When a design's wire load model is selected manually by the user (with the command **set_wire_load**), no wire load is selected automatically for that design.

To determine the current value of this variable use **printvar auto_wire_load_selection**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

`report_design(2)`

banking_enable_concatenate_name

Controls the multibit cell name used for the **create_register_bank** commands in the output file generated by the **set_banking_guidance_strategy** command.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Starting with the J-2014.09-SP4 release, the **banking_enable_concatenate_name** variable controls the name of the multibit registers created by the **create_register_bank** command in the output file generated by the **set_banking_guidance_strategy** command. By default, when the single-bit registers are grouped together, the tool uses the following naming style for multibit cells: The name of the original single-bit registers concatenated with an underscore (_), such as reg_1_reg_0.

For example, if the tool groups two single-bit registers, a_reg[0] and a_reg[1], to one multibit cell, it generates the following in the output file:

create_register_bank -name a_reg[0]_a_reg[1] {a_reg[0] a_reg[1]} -lib_cell MREG2

To revert to the naming style used in previous releases, set the **banking_enable_concatenate_name** variable to **false**. When the variable is **false**, the name of each register bank is created using the *prefixN1_N2* format, where N1 and N2 are integers. The N1 and N2 integers are used so the name is unique in the output file. The default prefix is group.

In the previous naming style, if the tool groups two single-bit registers, a_reg[0] and a_reg[1], to one multibit cell, the following command is generated in the output file:

create_register_bank -name group0_0 {a_reg[0] a_reg[1]} -lib_cell MREG2

You can change the name prefix in either naming style by specifying the **-name_prefix** option of the **set_banking_guidance_strategy** command.

For example, if you specify the MBIT prefix with the **-name_prefix** option using the default naming style, the output file generated during placement includes the following **create_register_bank** command:

create_register_bank -name MBIT_a_reg[0]_a_reg[1] {a_reg[0] a_reg[1]} -lib_cell MREG2

Use the following command to determine the current value of the variable:

```
prompt> printvar banking_enable_concatenate_name
```

SEE ALSO

`set_banking_guidance_strategy(2)`

bind_unused_hierarchical_pins

Specifies if unused hierarchical input pins should be connected to constant tie-off cells during compile. The unused hierarchical pin has neither load nor driver.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

The tool connects each undriven leaf cell input to a constant tie-off cell during compile. The tool also does the same for input pins of a design hierarchy. If an input pin of a design hierarchy is undriven, it connects to a constant tie-off cell in its parent hierarchy during compile.

To prevent unused hierarchical input pins from being tied off by the tool, set the **bind_unused_hierarchical_pins** variable to false before linking in the design.

When a command such as **compile -boundary_optimization** is used, the optimized design might be left with hierarchical inputs that are neither driven nor loaded. These hierarchical inputs are not connected to constant tie-offs if the **bind_unused_hierarchical_pins** variable is set to false.

To determine the current value of this variable, use the **printvar bind_unused_hierarchical_pins** command. For a list of all compile variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

bit_blasted_bus_linking_naming_styles

Specifies the bus pin naming styles to be matched.

TYPE

String

DEFAULT

%s[%d] %s(%d) %s_%d_

GROUP

system_variables

DESCRIPTION

When the **enable_bit_blasted_bus_linking** variable is set to true, the **bit_blasted_bus_linking_naming_styles** variable allows the linker to recognize bit-blasted buses by pin names during the link process. If the pin names follow a specific pattern, DC Explorer infers a bus when encountering the individual blasted bits.

Typically, the RTL pin names and the logic library pin names should match. Signals are defined the same way in both the RTL and the library. They are defined as buses or a bus is defined by its individual wires. Occasionally, mismatches occur during design development; for example, when you transfer the design from one technology to another. To fix the mismatches, you can set the **enable_bit_blasted_bus_linking** variable to true before the design is read.

Setting the **enable_bit_blasted_bus_linking** variable to true allows the linker to match buses and bit-blasted pin names according to the patterns specified by the **bit_blasted_bus_linking_naming_styles** variable. The default for the **enable_bit_blasted_bus_linking** variable is false.

For example,

```
prompt> set bit_blasted_bus_linking_naming_styles {%s[%d] %s_%d}
```

This setting permits the following U1 reference to link

```
input [1:0] in;
my_macro U1( .S(in),...
```

even if my_macro is defined as follows:

```
my_macro
input S_1;
input S_0;
\...
```

For multiple-dimension buses, the style can be explicitly defined or in certain cases implied. For example, the %s[%d] setting matches

A[0], A[0][0], or A[0][0][0], and so on.

However, to match the A_0__0__0_ pin name, you need to use the following setting:

%s_%d__%d__%d_

To determine the current setting of this variable, use the **printvar bit_blasted_bus_linking_naming_styles** command. For a list of all system variables and their current values, use **print_variable_group system**.

SEE ALSO

`enable_bit_blasted_bus_linking(3)`

bound_attributes

Contains attributes related to bound.

DESCRIPTION

Contains attributes related to bound.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class bound -application**, the definition of attributes can be listed.

Bound Attributes

aspect_ratio

Specifies the **width**:**height** ratio of a bound.

The data type of **aspect_ratio** is double.

This attribute is read-only.

bbox

Specifies the bounding-box of a bound. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ll

Specifies the lower-left corner of the bounding-box of a bound.

The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ll** of a bound, by accessing the first element of its **bbox**.

The data type of **bbox_ll** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a bound.

The data type of **bbox_llx** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a bound.

The data type of **bbox_lly** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ur

Specifies the upper-right corner of the bounding-box of a bound.

The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ur** of a bound, by accessing the second element of its **bbox**.

The data type of **bbox_ur** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a bound.

The data type of **bbox_urx** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a bound.

The data type of **bbox_ury** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

color

Specifies color to draw a move bound and its associated instances.

The data type of **color** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

dimension

Specifies dimension of a group bound.

Its format is {**width height**}.

The data type of **dimension** is string.

This attribute is read-only.

effort

Specifies effort to bring cells closer inside an auto group bound.

Its valid values can be:

- **low**
- **medium**
- **high**
- **ultra**

The data type of **effort** is string.

This attribute is read-only.

is_mhr

Specifies whether the exclusive movebound is a multi-height row region.

The data type of **is_mhr** is boolean.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name

Specifies name of a bound object.

The data type of **name** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

number_of_hard_macro

Specifies number of hard macro cells inside a bound.

The data type of **number_of_hard_macro** is integer.

This attribute is read-only.

number_of_standard_cell

Specifies number of standard cells inside a bound.

The data type of **number_of_standard_cell** is integer.

This attribute is read-only.

object_class

Specifies object class name of a bound, which is **bound**.

The data type of **object_class** is string.

This attribute is read-only.

object_type

Specifies object type of a bound, which can be **move_bound**, **auto_group_bound**, or **group_bound**.

The data type of **object_type** is string.

This attribute is read-only.

points

Specifies point list of a move bound's boundary.

The data type of **points** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

type

Specifies type of a move bound or a group bound.

The data type of **type** is string.

Its valid values are:

soft

hard

exclusive

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

utilization

Specifies the ratio of total area size of associated instances to the area of a move bound.

The data type of **utilization** is double.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

budget_auto_limit_load

Controls the command *allocate_fp_budgets* to apply the load limitation of a buffer to the pin load, wire load, and fanout number constraints it generates.

TYPE

binary

DEFAULT

false

GROUP

budgeting_variables

DESCRIPTION

This variable controls command *allocate_fp_budgets* to apply the load limitation of a buffer to the pin load, wire load, and fanout number constraints it generates. The buffer it uses is a medium sized buffer in the design reference libraries. The affected constraints are: *set_port_fanout_number set_load -pin_load -max set_load -pin_load -min set_load -wire_load -max set_load -wire_load -min*

If this variable is not set or is set to false, it means no such limit will be applied.

There are user controls to the value of these limits through variables: budget_port_fanout_number, budget_max_pin_load, budget_min_pin_load, budget_max_wire_load, and budget_min_wire_load. These manual controls overwrite the values the tool finds from a buffer.

SEE ALSO

```
allocate_fp_budgets(2)
budget_port_fanout_number(3)
budget_max_pin_load(3)
budget_min_pin_load(3)
budget_max_wire_load(3)
budget_min_wire_load(3)
```

budget_generate_critical_range

Enables automatic generation of **set_critical_range** commands by **dc_allocate_budgets** for multiply-instantiated subdesigns.

TYPE

Boolean

DEFAULT

false

GROUP

acs_variables

DESCRIPTION

When set to true, this variable causes the **dc_allocate_budgets** command to automatically generate a **set_critical_range** command for any budgeted cells that have multiply-instantiated subdesigns. The value of **critical_range** is set to 10% of the shortest clock period in the design. If the top-level design already has a **critical_range** attribute, then that original value is used instead.

The purpose of this is to improve QoR, since any multiple instances must be dont_touched after compile in a bottom-up compile flow. Using **set_critical_range** causes any near-critical paths in these blocks to be optimized before the block is dont_touched.

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

```
prompt> printvar budget_generate_critical_range
```

SEE ALSO

budget_map_clock_gating_cells

Maps integrated clock gating cells into target library cells during RTL budgeting.

TYPE

Boolean

DEFAULT

false

GROUP

acs_variables

DESCRIPTION

When set to true, this variable causes the **dc_allocate_budgets -mode rtl** command to map any integrated clock gating cells into target library cells before calculating the budget. Normally, any unmapped cells would remain unmapped during budgeting when **-mode rtl** is used.

The purpose of this is to prevent incorrect **-level_sensitive** delay constraints from appearing in the budget constraint files. These may otherwise appear because unmapped integrated clock gating cells can include transparent latch elements.

To see the current value of this variable, use the following command:

```
prompt> printvar budget_map_clock_gating_cells
```

SEE ALSO

budget_max_pin_load

Specifies the maximum load value for the *set_load -pin_load -max* constraint generated by budgeting.

TYPE

float

DEFAULT

0

GROUP

budgeting_variables

DESCRIPTION

This variable specifies the maximum load value for the *set_load -pin_load -max* constraint budgeting generates for a block pin. If this variable is not set or is set to 0.0, it means there is no limit.

SEE ALSO

```
allocate_fp_budgets(2)
budget_auto_limit_load(3)
```

budget_max_wire_load

Specifies the maximum load value for the *set_load -wire_load -max* constraint generated by budgeting.

TYPE

float

DEFAULT

0

GROUP

budgeting_variables

DESCRIPTION

This variable specifies the maximum load value for the *set_load -wire_load -max* constraint budgeting generates for a block pin. If this variable is not set or is set to 0.0, it means there is no limit.

SEE ALSO

```
allocate_fp_budgets(2)
budget_auto_limit_load(3)
```

budget_min_pin_load

Specifies the maximum load value for the *set_load -pin_load -min* constraint generated by budgeting.

TYPE

float

DEFAULT

0

GROUP

budgeting_variables

DESCRIPTION

This variable specifies the maximum load value for the *set_load -pin_load -min* constraint budgeting generates for a block pin. If this variable is not set or is set to 0.0, it means there is no limit.

SEE ALSO

```
allocate_fp_budgets(2)
budget_auto_limit_load(3)
```

budget_min_wire_load

Specifies the maximum load value for the *set_load -wire_load -min* constraint generated by budgeting.

TYPE

float

DEFAULT

0

GROUP

budgeting_variables

DESCRIPTION

This variable specifies the maximum load value for the *set_load -wire_load -min* constraint budgeting generates for a block pin. If this variable is not set or is set to 0.0, it means there is no limit.

SEE ALSO

```
allocate_fp_budgets(2)
budget_auto_limit_load(3)
```

budget_port_fanout_number

Specifies the maximum number of fanouts for set_port_fanout_number constraint generated by budgeting.

TYPE

integer

DEFAULT

0

GROUP

budgeting_variables

DESCRIPTION

This variable specifies the maximum number of fanouts for the set_port_fanout_number constraint budgeting generates for a block pin. If this variable is not set or is set to 0, it means there is no limit.

SEE ALSO

```
allocate_fp_budgets(2)
budget_auto_limit_load(3)
```

budgeting_allow_negative_delays

Uses to control the presence of negative input and output delays during budgeting

TYPE

Boolean

DEFAULT

false

GROUP

budgeting_variables

DESCRIPTION

During budgeting, when the values of the computed input or output delays are negative, they are rounded up to zero by default. When this variable is set to true, the negative input or output delays are printed.

SEE ALSO

`allocate_fp_budgets(2)`

budgeting_enable_hier_si

Controls the propagation of hierarchical signal integrity information to the blocks during budgeting

TYPE

Boolean

DEFAULT

false

GROUP

budgeting_variables

DESCRIPTION

When the variable is set to true, budgeting computes signal integrity information that can affect block implementation and passes this information to the blocks. Use this feature only with designs that have detailed routes on the top levels.

SEE ALSO

`allocate_fp_budgets(2)`

bus_inference_descending_sort

Specifies that the members of that port bus are to be sorted in descending order rather than in ascending order.

TYPE

Boolean

DEFAULT

true

GROUP

edif_variables
io_variables

DESCRIPTION

Affects the **read** command except for the db, Verilog and VHDL formats. This variable is primarily used when reading in designs in the LSI/NDL format. That particular format does not support representation of busses, but, if port names follow a specific pattern (as described in the variable **bus_inference_style**), the individual bits can be "inferred" into a port bus. When true (the default value), This variable specifies that the members of that port bus are to be sorted in descending order rather than in ascending order.

For example, with the variable **bus_inference_style** set to "%s[%d]", the ports "A[1]", "A[2]", "A[3]", and "A[4]" will be "inferred" into a port bus named "A". If this variable is true, the port bus "A" will have an index from 4 to 1; if this variable is false, the port bus "A" will have an index from 1 to 4.

With the variable **bus_inference_style** set to "#%d%%%s", the ports "#8%cb", "#9%cb", "#10%cb", and "#11%cb" will be "inferred" into a port bus named "cb". If this variable is true, the port bus "cb" will have an index from 11 to 8; if this variable is false, the port bus "cb" will have an index from 8 to 11.

To determine the current value of this variable, type **printvar bus_inference_descending_sort**. For a list of all **edif** or **io** variables and their current values, type **print_variable_group edif** or **print_variable_group io**.

SEE ALSO

remove_bus(2)
report_bus(2)
bus_inference_style(3)
bus_minus_style(3)
bus_naming_style(3)
bus_range_separator_style(3)
io_variables(3)

bus_inference_style

Specifies the pattern used to infer individual bits into a port bus.

TYPE

string

DEFAULT

""

GROUP

edif_variables
io_variables

DESCRIPTION

This variable specifies the pattern used to infer individual bits into a port bus. The variable affects the **read** command except for the .db and VHDL formats. This variable also affects the VHDL **write** commands. The variable is used primarily when reading in designs in the LSI/NDL format. The LSI/NDL format does not support representation of buses. But if port names follow a specific pattern (as described by this variable), the individual bits can be inferred into a port bus. If you specify an invalid value, no port buses are inferred.

When running in Tcl mode, the **bus_inference_style** value must be specified within curly brackets ({}). For example:

```
set bus_inference_style {%s[%d]}
```

This variable must contain 1 %s (percent s) and %d (percent d) character sequence. Additional characters can be used with these symbols. To use a percent sign in a name, 2 percent signs are needed in the variable string (%%).

In naming a port bus, the port name is substituted for %s, and the port number replaces %d. A single percent sign is substituted for %%.

For example, with this variable set to "%s[%d]", the ports "A[1]", "A[2]", "A[3]", and "A[4]" will be inferred into a port bus named A either with an index from 1 to 4, or with an index from 4 to 1 (see **bus_inference_descending_sort**).

With this variable set to "#%d%%%s", the ports "#8%cb", "#9%cb", "#10%cb", and "#11%cb" will be inferred into a port bus named cb either with an index from 8 to 11, or with an index from 11 to 8 (see **bus_inference_descending_sort**).

To determine the current value of this variable, use the **printvar bus_inference_style** command. For a list of all **edif** or **io** variables and their current values, use the **print_variable_group edif** or **print_variable_group io** command.

SEE ALSO

```
remove_bus(2)
report_bus(2)
bus_inference_descending_sort(3)
bus_minus_style(3)
bus_naming_style(3)
bus_range_separator_style(3)
io_variables(3)
```

bus_minus_style

Controls the naming of individual members of bit-blasted port, instance, or net buses with negative indices.

TYPE

string

DEFAULT

-%d

GROUP

hdl_variables

DESCRIPTION

This variable controls the naming of individual members of bit-blasted port, instance, or net buses with negative indices. The variable affects the **read** command with the **vhdl** format option.

To determine the current value of this variable, use the **printvar bus_minus_style** command. For a list of all HDL variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

```
remove_bus(2)
report_bus(2)
bus_inference_descending_sort(3)
bus_inference_style(3)
bus_naming_style(3)
bus_range_separator_style(3)
```

bus_multiple_name_separator_style

Determines the separator used to name of a multibit cell that implements bits whose original base_names differ.

TYPE

string

DEFAULT

,,

GROUP

multibit_variables

DESCRIPTION

This variable affects the naming of multibit cells during multibit mapping. The variable is used to name a multibit cell whose original single-bit cells had different base names. The default value is a double **bus_multiple_separator_style** pattern, and is used if an invalid value is specified (or no value at all).

The **bus_multiple_separator_style** variable is used to separate two ranges of bits sharing the same base names, while **bus_multiple_name_separator_style** is used to separate the different base names.

The two variables are used in conjunction with the **bus_naming_style** and the **bus_range_separator_style** variable to generate names for multibit cells.

In the following examples assume that **bus_range_separator_style** is set to ":", **bus_multiple_name_separator_style** is set to "_MB_" and **bus_multiple_separator_style** is set to ",".

If cells with the names q_1[0], q_1[1], q_2[2], q_2[5], q_2[6], and q_2[7] are packed into a 6-bit wide cell, the name given to the new cell is q_1[0:1]_MB_q_2[2,5:7].

If cells q[0], q[2], q[4], and q[6] are packed into a 4-bit wide cell, the name given to the new cell is q[0,2,4,6].

To determine the current value of this variable, use the following command: **printvar bus_multiple_name_separator_style**.

For a list of all **multibit** variables and their current values, use: **print_variable_group multibit**

SEE ALSO

`bus_multiple_separator_style(3)`
`bus_range_separator_style(3)`

bus_multiple_separator_style

Determines the name of a multibit cell that implements bits that do not form a range.

TYPE

string

DEFAULT

,

GROUP

multibit_variables

DESCRIPTION

This variable affects the naming of multibit cells during multibit mapping. The variable is used to name a multibit cell that implements bits that do not form a range. The default is used if an invalid value is specified.

The **bus_range_separator_style** variable is used to separate the start and end bit positions of a range, while **bus_multiple_separator_style** is used to separate two ranges. The two variables are used in conjunction with the **bus_naming_style** variable to generate names for multibit cells.

Assume that **bus_range_separator_style** is set to ":", **bus_multiple_separator_style** is set to ",", and **bus_naming_style** is set to "%s[%d] in the following examples.

For example, if cells with the names q[0], q[1], q[2], q[5], q[6], and q[7] are packed into a 6-bit wide cell, the name given to the new cell is q[0:2,5:7]. If cells q[0], q[2], q[4], and q[6] are packed into a 4-bit wide cell, the name given to the new cell is q[0,2,4,6].

To determine the current value of this variable, use the **printvar bus_range_separator_style** command. For a list of all **multibit** variables and their current values, use the **print_variable_group multibit** command.

SEE ALSO

```
bus_range_separator_style(3)
```

bus_naming_style

Specifies the style to use in naming an individual port member, net member, or cell instance member of an EDIF array or of a Verilog or VHDL vector.

TYPE

string

DEFAULT

%s[%d]

GROUP

```
edif_variables
hdl_variables
schematic_variables
```

DESCRIPTION

This variable affects the **read** command with the EDIF, Verilog, or VHDL format option, the **write** command with the EDIF format option, and the **create_schematic** command with the busing option.

When reading buses, this variable specifies the style to use in naming an individual port member, net member, or cell instance member of an EDIF array or of a Verilog or VHDL vector.

When running in TCL mode the **bus_naming_style** needs to be enclosed by curly brackets. For example:

```
set bus_naming_style {%s[%d]}
```

When writing buses, this variable used with the **bus_range_separator_style** variable specifies the style to use in naming a port array or net array in the EDIF file. If you specify an invalid value, the array is given the name of the bus. When writing schematic nets, this variable used with the **bus_range_separator_style** variable specifies the style to use in naming a net connected to the "wire" end of a ripper in the EDIF file. If you specify an invalid value, the net is given the name of the original net.

When creating schematics, this variable used with the variable **bus_range_separator_style** variable specifies the style to use in naming a ripper, bused port, bused net, or net connected to the "wire" end of a ripper. If you specify an invalid value, the default value is used.

This variable must contain only 1 %s (percent s) and 1 %d (percent d) character sequence. To use the % (percent sign) in the name, use 2 of them in the variable string (%%). Therefore, the only characters that can follow a percent sign are %, s, or d.

When reading buses, in naming members, the name of the array is substituted for %s, and the number of the member is substituted for %d. One percent sign is substituted for %%.

For example, if this variable is set to "%s[%d]", then the first member of the 4-bit array "A", going from 0 to 3, is named:

```
A[0]
```

If this variable is set to "%s_%%%d.X", then the first member of the 8-bit array "xy", going from 9 to 16, is named:

```
xy_%9.X
```

See the **bus_dimension_separator_style** man page for a description of how it is used in conjunction with this variable for specifying the names of the members of multidimensional arrays in the EDIF format or multidimensional vectors in the VHDL format.

See the **bus_minus_style** man page for a description of how to specify the names of vectors with negative indices in the VHDL format.

When creating schematics or when writing buses, in naming a bused port or bused net or a port array or net array, the original bus or array name is substituted for %s. The start and end bits of the bus or array separated by the value of the variable **bus_range_separator_style** are substituted for %d. One percent sign is substituted for each %%.

For example, if this variable is set to "%s[%d:%d]", then the 4-bit bus or array "A", going from 0 to 3, is named:

```
A[0:3]
```

If this variable is set to "%%%s[%d][%d]", then the 8-bit bus or array "B", going from -4 to 3, is named:

```
%B[-4][3]
```

See the **edifout_multidimension_arrays** man page for a description of how it is used in conjunction with this variable for specifying the names of multidimensional arrays.

See the **edifout_numerical_array_members** man page for a description of how it is used in conjunction with this variable for specifying the names of descending arrays.

When creating schematics or when writing schematic nets, in naming a ripper or net connected to the "wire" end of a ripper, the original net name is substituted for %s. If the net is a scalar (a single bit) net, the bit of the ripper is substituted for %d. If the net is a bused net, the start and end bits of the ripper separated by the value of the variable **bus_range_separator_style** are substituted for %d. One percent sign is substituted for each %%.

For example, if this variable is set to "%s[%d]", then the ripper or the net connected to the "wire" end of the ripper that is ripping off the first bit of the 4-bit net array "A" going from 0 to 3, is named:

```
A[0]
```

If this variable is set to "%s_%%%d.X", then the ripper or the net connected to the "wire" end of the ripper that is ripping off the first bit of the 8-bit net array "xy" going from 9 to 16, is named:

```
xy_%9.X
```

If this variable is set to "%s[%d]" and the **bus_range_separator_style** variable is set to ":", then the net connected to the "wire" end of the ripper that is ripping off the third through fifth bits of the 8-bit net array "xy" going from 9 to 16, is named:

```
xy[11:13]
```

If this variable is set to "%s_%%%d.X" and the **bus_range_separator_style** variable is set to "..", then the ripper or the net connected to the "wire" end of the ripper that is ripping off the third through fourth bits of the 4-bit net array "A" going from 0 to 3, is named:

```
A_%2..3.X
```

To determine the current value of this variable, use the **printvar bus_naming_style** command. For a list of all EDIF, HDL, or schematic variables and their current values, use the **print_variable_group edif**, **print_variable_group hdl**, or **print_variable_group schematic** command.

SEE ALSO

```
remove_bus(2)
report_bus(2)
bus_inference_descending_sort(3)
bus_inference_style(3)
bus_minus_style(3)
bus_range_separator_style(3)
```

bus_range_separator_style

Specifies the style to use in naming a net connected to the "wire" end of a ripper in the EDIF file.

TYPE

string

DEFAULT

:

GROUP

```
edif_variables
schematic_variables
```

DESCRIPTION

This variable specifies the style to use in naming a net connected to the "wire" end of a ripper in the EDIF file. This variable affects the **write** command with the EDIF format option and the **create_schematic** command with the busing option.

When writing buses, this variable used with the **bus_naming_style** variable, specifies the style to use in naming a port array or net array in the EDIF file. When writing schematic nets, this variable used with the **bus_naming_style** variable, specifies the style to use in naming a net connected to the "wire" end of a ripper in the EDIF file.

When creating schematics, this variable used with the **bus_naming_style** variable, specifies the style to use in naming a ripper, bused port, bused net, or net connected to the "wire" end of a ripper. If you specify an invalid value, the default value is used.

See the **bus_naming_style** man page for a description of how this variable is used in conjunction with that variable.

To determine the current value of this variable, use the **printvar bus_range_separator_style** command. For a list of all EDIF or schematic variables and their current values, use the **print_variable_group edif** or **print_variable_group schematic** command.

SEE ALSO

```
remove_bus(2)
report_bus(2)
bus_inference_descending_sort(3)
bus_inference_style(3)
bus_minus_style(3)
bus_naming_style(3)
```

cache_dir_chmod_octal

Specifies the value of the mode bits for created cache directories.

TYPE

string

DEFAULT

777

GROUP

synlib_variables

DESCRIPTION

Cache directories are created with their mode bits set to the value of the **cache_dir_chmod_octal** variable. The value of this variable is a string that is translated to an octal number. There are separate variables for directories and files to allow the sticky bit to be set.

Many UNIX systems allow a sticky bit to be set on directories. Note that setting the sticky bit on files has a different meaning and is not allowed for cache files. If the sticky bit on a directory is set, and if you have write permission on the directory, then you can write your own files in the directory, but you cannot delete the files of other people in that directory. See the UNIX man page on sticky(8) for more information.

Caches are often shared among users, and the sticky bit allows users to write into the same directory without worrying that other users will overwrite their files. If your UNIX system does not have sticky bit capabilities, your system administrator should remove that bit from the **cache_dir_chmod_octal** default in the system .synopsys_dc.setup file.

SEE ALSO

```
cache_ls(1)
cache_file_chmod_octal(3)
```

cache_file_chmod_octal

Specifies the value of the mode bits for created cache files.

TYPE

string

DEFAULT

666

GROUP

synlib_variables

DESCRIPTION

Cache files are created with their mode bits set to the value of **cache_file_chmod_octal**. The value of this variable is a string that is translated to an octal number. Cache directories use the **cache_dir_chmod_octal** variable to set the mode bits. There are separate variables for directories and files to allow the sticky bit to be set.

SEE ALSO

```
cache_ls(1)
cache_dir_chmod_octal(3)
```

case_analysis_log_file

Specifies the name of a log file generated during propagation of constant values, from case analysis or from nets tied to logic zero or logic one. Each scenario has its proprietary log file if multiple scenarios exist.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable specifies the name of a log file generated during the propagation of constant values, from case analysis or from nets tied to logic zero or logic one. The log file contains the list of all ports 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.

In multicorner-multimode, you must specify the log file name in the definition of each scenario, or no log will be generated for that scenario. If you switch the active scenario, the log file used will be switched automatically. Log files for different scenarios can have the same name.

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

To determine the current value of this variable, use the **printvar case_analysis_log_file** command. Note that if you switch the scenario in multicorner-multimode, the value of this variable remains at the value last set.

SEE ALSO

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

disable_case_analysis(3)

case_analysis_propagate_through_icg

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

TYPE

Boolean

DEFAULT

false

DESCRIPTION

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

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

To activate logic propagation through all integrated clock gating cells, you must set the following:

set case_analysis_propagate_through_icg true

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

SEE ALSO

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

case_analysis_sequential_propagation

Determines whether case analysis is propagated across sequential cells.

TYPE

string

DEFAULT

never

DESCRIPTION

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

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

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

```
printvar case_analysis_sequential_propagation

echo $case_analysis_sequential_propagation
```

SEE ALSO

```
printvar(2)
set_case_analysis(2)
case_analysis_propagate_through_icg(3)
```

case_analysis_with_logic_constants

Enables constant propagation when set to true, even if a design contains only logic constants.

TYPE

Boolean

DEFAULT

true

GROUP

timing

DESCRIPTION

When set to true (the default), this variable enables constant propagation, even if a design contains only logic constants. When set to false, constant propagation is not performed unless a **set_case_analysis** command is specified. The **disable_case_analysis** variable overrides the **case_analysis_with_logic_constants** variable. If the **disable_case_analysis** variable is set, no constants are propagated.

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

SEE ALSO

```
remove_case_analysis(2)
report_case_analysis(2)
set_case_analysis(2)
disable_case_analysis(3)
```

cell_attributes

Contains attributes that can be placed on a cell.

DESCRIPTION

Contains attributes that can be placed on a cell.

There are a number of commands used to set attributes, however, most attributes can be set with the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute. Otherwise, use **set_attribute**. If an attribute is read-only, you cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Cell Attributes

allowable_orientation

Specifies the allowable orientation list on a cell.

This attribute may exist on standard cell, macro cell and cover cell.

This attribute is read-only and cannot be modified.

area

Specifies the area of a cell. This attribute does not exist on a hierarchical cell.

The tool calculates the attribute by the cell's boundary points.

This attribute is read-only and cannot be modified.

aspect_ratio

Specifies the **height**:**width** ratio of a cell. This attribute does not exist on a hierarchical cell.

This attribute is read-only and cannot be modified.

async_set_reset_q

Establishes the value (0 or 1) that should be assigned to the q output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_qn**. Use

these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.

Note: If you are unsure whether or not your technology library uses V3.0a syntax, ask your ASIC vendor.

async_set_reset_qn

Establishes the value (0 or 1) that should be assigned to the qn output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_q**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then by default if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.

Note: If you are unsure whether or not your technology library uses V3.0a syntax, ask your ASIC vendor.

disable_timing *

Disables the timing arcs of a cell. This has the same effect on timing as not having the arc in the library. Set with the **set_disable_timing** command.

dont_touch

Identifies cells to be excluded from optimization. Values are **true** (the default) or **false**. Cells with the **dont_touch** attribute set to **true** are not modified or replaced during **compile**. Setting **dont_touch** on a hierarchical cell sets the attribute on all cells below it. Set with **set_dont_touch**.

eco_change_status

Shows the current ECO status of a leaf cell instance, including the instance is changed by which netlist editing command, is legalized or not.

This attribute is settable and removable.

eco_status

Indicate the leaf cell ECO status for freeze silicon flow.

This attribute is settable but not removable.

flip_flop_type

Stores the name of the specified flip-flop to be converted from the **target_library**. The **compile** command automatically converts all tagged flip-flops to the specified (or one similar) type. Set with **set_register_type -flip_flop** *flip_flop_name [cell_list]*.

flip_flop_type_exact

Stores the name of the specified flip-flop to be converted from the **target_library**. The **compile** command automatically converts all tagged flip-flops to the exact flip-flop type. Set with **set_register_type -exact -flip_flop** *flip_flop_name [cell_list]*.

height

Specifies the height of a cell. This attribute does not exist on a hierarchical cell.

The tool uses the cell's cell boundary to calculate its height.

This attribute is read-only and cannot be modified.

is_black_box

Set to **true** if the cell's reference is not linked to a design.

This attribute is read-only and cannot be modified.

is_block_abstraction

Set to *true* if the cell is linked to block abstraction

This attribute is read-only and cannot be modified.

is_clock_gating_check

Set and/or reset by the identify_clock_gating command to indicate that the cell is part of the clock gating structure.

This attribute is read-only and cannot be modified except by the identify_clock_gating command.

is_combinational

Set to **true** if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box.

This attribute is read-only and cannot be modified.

is_dw_subblock

Set to **true** if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated.

This attribute is read-only and cannot be modified.

Note: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical

Set to **true** if the design contains leaf cells or other levels of hierarchy.

This attribute is read-only and cannot be modified.

is_ilm

Set to *true* if the cell is linked to an ILM

This attribute is read-only and cannot be modified.

is_mapped

Set to **true** if the cell is not generic logic.

This attribute is read-only and cannot be modified.

is_mim

Specifies that a soft macro is multiply instantiated module (MIM).

This attribute is read-only and cannot be modified.

is_mim_master_instance

Specifies that a soft macro is a master instance among a set of multiply instantiated modules (MIMs).

This attribute is read-only and cannot be modified.

is_sequential

Set to **true** if the cell is sequential. A cell is sequential if it is not combinational.

This attribute is read-only and cannot be modified.

is_synlib_module

Set to **true** if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator.

This attribute is read-only and cannot be modified.

Note: synlib modules that are manually elaborated will not have this attribute.

is_synlib_operator

Set to **true** if the object (a cell or a reference) is a synthetic library operator reference.

This attribute is read-only and cannot be modified.

is_test_circuitry

Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry.

This attribute is read-only and cannot be modified.

is_unmapped

true if the cell is generic logic.

This attribute is read-only and cannot be modified.

latch_type_exact

Stores the name of the specified latch to be converted from the **target_library**. The **compile** command automatically converts all tagged latches to the exact latch type. Set with **set_sequential_type -latch** *latch_name [cell_list].*

macro_area_percentage

Specifies the percentage of total macro area of a soft macro's physical area.

This attribute is read-only and cannot be modified.

macro_internal_switch_pg_pins

Specifies the name list of macro internal switch PG pins, which will be used by connect_supply_net to connect internal supply net with.

This attribute is modifiable and it's only for macro cells with internal switches.

map_only

When set to **true**, **compile** will attempt to map the object exactly in the target library, and will exclude the object from logic-level optimization (flattening and structuring). The default is **false**. Set with **set_map_only**.

mask_layout_type

Specifies the mask layout type of a cell.

This attribute is read-only and cannot be modified.

max_fall_delay

A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_metal_layer

Specifies the reserved maximum metal layer name of a soft macro or a black box.

This attribute is read-only and cannot be modified.

max_rise_delay

A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow

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 technology library. Set with **set_max_time_borrow**.

mim_master_name

Specifies the name of the master for a multiply instantiated module (MIM) soft macro.

This attribute is read-only and cannot be modified.

min_fall_delay

A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_metal_layer

Specifies the reserved minimum metal layer name of a soft macro or a black box.

This attribute is read-only and cannot be modified.

min_rise_delay

A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

number_of_black_box

Specifies the count of black boxes in a hierarchical cell or a soft macro.

Whether a cell is a black box can be determined by the attribute is_black_box.

This attribute is read-only and cannot be modified.

number_of_io_cell

Specifies the count of io cells in a hierarchical cell or a soft macro.

The tool counts cells in when its mask_layout_type is **io_pad**, **corner_pad**, **pad_filler**, or **flip_chip_pad**.

This attribute is read-only and cannot be modified.

number_of_macro

Specifies the count of macros in a hierarchical cell or a soft macro.

The tool counts cells in when its mask_layout_type is macro.

This attribute is read-only and cannot be modified.

number_of_pinshape

Specifies the number of pin shapes of a soft macro.

This attribute is read-only and cannot be modified.

number_of_standard_cell

Specifies the count of standard cells in a hierarchical cell or a soft macro.

The tool counts cells in when its **mask_layout_type** matches ***std***.

This attribute is read-only and cannot be modified.

orientation

Specifies the orientation of a cell.

This attribute does not exist on a hierarchical cell.

This attribute can be modified by set_attribute. When a cell has allowable orientation list while you set its orientation which is not in the list of cell::allowable_orientation, you will get a warning message.

physical_area

Specifies the physical area of a hierarchical cell or a soft macro.

The physical area of a hierarchical cell is the sum of its direct children's **physical_area** or **area**. If a direct child is a hierarchical cell, then **physical_area** is used. If a child is a standard cell or macro, then **area** is used, otherwise that child is skipped.

The physical area of a soft macro is the sum of its children's **physical_area** or **area**. The children will be iterated from the subdesign file. If a child is a soft macro, then **physical_area** is used, if a child is a standard cell or hard macro, **area** is used, otherwise that child is skipped.

This attribute is read-only and cannot be modified.

physical_area_percentage_in_top_design

Specifies the percentage of **physical_area** of a soft macro in the top design.

This attribute is read-only and cannot be modified.

ref_name

The reference name of a cell.

This attribute is read-only and cannot be modified.

scan

When **true**, specifies that the cell is always replaced by an equivalent scan cell during **insert_dft**. When **false**, the cell is not replaced. Set with **set_scan**.

scan_chain

Includes the specified cells of the referenced design in the scan-chain whose index is the value of this attribute. Set with **set_scan_chain**.

test_dont_fault

Specifies cells not faulted during test pattern generation. If no command options are specified, this attribute is set for both "stuck-at-0" and "stuck-at-1" faults. Set with **set_test_dont_fault**.

test_isolate

Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.

Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

test_routing_position

Specifies the preferred routing order of the scan-test signals of the identified cells. Set with **set_test_routing_order**.

ungroup

Removes a level of hierarchy by exploding the contents of the specified cell in the current design. If specified on a reference object, cells using that reference are ungrouped during **compile**. Set with **set_ungroup**.

utilization

Specifies the utilization of a soft macro.

The tool calculates the utilization using **physical_area**:**area** ratio of a soft macro.

This attribute is read-only and cannot be modified.

width

Specifies the width of a cell. This attribute does not exist on a hierarchical cell.

The tool uses the cell's cell boundary to calculate its width.

This attribute is read-only and cannot be modified.

within_block_abstraction

Specifies whether the cell is part of the block abstraction.

This attribute is read-only and cannot be modified.

within_ilm

Specifies whether the cell is part of an ILM.

This attribute is read-only and cannot be modified.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)
```

cell_site_attributes

Contains attributes related to cell site.

DESCRIPTION

Contains attributes related to cell site.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class cell_site -application**, the definition of attributes can be listed.

Cell Site Attributes

bbox

Specifies the bounding-box of a cell site. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is read-only.

constraints

Specifies the maximum number of flip chip driver of the particular personality type that can be placed in a flip chip driver island.

The data type of **constraints** is string.

This attribute is read-only.

layer

Specifies system layer name on which cell site is.

The data type of **layer** is string.

This attribute is read-only.

object_class

Specifies object class name of a cell site, which is **cell_site**.

The data type of **object_class** is string.

This attribute is read-only.

orientations

Specifies allowable orientations of flip chip driver when placed.

The data type of **orientations** is string.

This attribute is read-only.

personality

Specifies personality types of flip-chip driver cells to be placed in a cell site.

The data type of **personality** is string.

This attribute is read-only.

reserved_slots

Specifies that certain locations pointed by the row and column indices in an flip chip driver island are reserved for a certain flip chip driver cell. Only the specified flip chip driver can be placed in the locations.

Its format is like, {{**lib_cell_name** {**col row**}...} ...}

The data type of **reserved_slots** is string.

This attribute is read-only.

rotate_by_row

Specifies forced orientations of the drivers are set by rows.

The data type of **rotate_by_row** is boolean.

This attribute is read-only.

style

Specifies the style to place flip chip drivers.

Its valid values can be:

- **flip_chip_array**
- **flip_chip_island**
- **flip_chip_ring**
- **flip_chip_row**

- **flip_chip_strip**

The data type of **style** is string.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
set_flip_chip_driver_array(2)
set_flip_chip_driver_island(2)
set_flip_chip_driver_ring(2)
```

change_names_bit_blast_negative_index

Bit-blast the bus if any bit of it is negative.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

When this variable is set to true, **change_names** bit-blasts the bus if any bit is negative. Otherwise, **change_names** shifts the negative range to the positive range starting at 0. The default value is false.

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

SEE ALSO

```
change_names(2)
define_name_rules(2)
```

change_names_dont_change_bus_members

Controls how the **change_names** command modifies the names of bus members.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

This variable is for the **change_names** command, and affects bus members only of bused ports or nets. When false (the default), **change_names** gives bus members the base name from their owning bus. For example, if BUS A has range 0 to 1 with the first element NET1 and the second element NET2, **change_names** changes NET1 to A[0] and NET2 to A[1]. When this variable is set to true, **change_names** does not change the names of bus members, so that NET1 and NET2 remain unchanged.

This variable also applies to **-special** rules, but has no effect if the name is changed by other rules that have higher priority than **-special** when **-special** rules are used; such as **-equal_ports_nets** and **-case_insensitive**. For more information, see the APPLYING NAME RULES section of the **define_name_rules** man page.

To determine the current value of this variable, use the **printvar change_names_dont_change_bus_members** command. For a list of all system variables and their current values, use the **print_variable_group system** command.

SEE ALSO

```
change_names(2)
define_name_rules(2)
system_variables(3)
```

check_design_allow_multiply_driven_nets_by_inputs_and_outputs

Controls whether the tool checks for input ports that connect to multiply driven nets whose drivers include an output port or pin.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

When this variable is set to **true**, the **check_design** command skips the checking of input ports that connect to multiply driven nets whose drivers include an output port or pin.

When the variable is set to **false**, the tool reports errors when input ports are connected to multiply driven nets whose drivers include an output port or pin. When the **check_design** command encounters an error, it continues to run without any change in return status.

To determine the current value of this variable, use the **get_app_var check_design_allow_multiply_driven_nets_by_inputs_and_outputs** command.

SEE ALSO

`check_design(2)`

check_design_allow_non_tri_drivers_on_tri_bus

Specifies the severity level to apply when the tool finds three-state buses with non-three-state drivers in the design.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

When this variable is set to true, the **check_design** command issues warnings when three-state buses with non-three-state drivers exist in the design. The command continues running after a warning is issued.

When the variable is set to false, the **check_design** command reports errors on three-state buses. When the **check_design** command encounters an error, it continues to run without any change in return status.

The default value of this variable is true.

To determine the current value of this variable, use the **get_app_var check_design_allow_non_tri_drivers_on_tri_bus** command.

SEE ALSO

`check_design(2)`

check_design_allow_unknown_wired_logic_type

Specifies the severity level to apply when the tool finds nets with multiple drivers of the unknown wired-logic type.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

When this variable is set to **true**, the **check_design** command issues warnings when nets with multiple drivers of the unknown wired-logic type exist in the design. The command continues running after issuing the warning.

When the variable is set to **false**, the **check_design** command reports errors on nets with multiple drivers of the unknown wired-logic type. When the **check_design** command encounters an error, it continues to run without any change in return status.

The default value of this variable is **true**.

To determine the current value of this variable, use the **get_app_var check_design_allow_unknown_wired_logic_type** command.

SEE ALSO

`check_design(2)`

check_design_check_for_wire_loop

Controls whether the tool checks for wire loops that have a timing loop with no cells in it.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

When this variable is set to true, the **check_design** command issues an error when wire loops are detected in the design. When the **check_design** command encounters an error, it continues to run without any change in return status.

When the variable is set to false, the tool skips the checking of wire loops in the design.

The default value of this variable is true.

To determine the current value of this variable, use the **get_app_var check_design_check_for_wire_loop** command.

SEE ALSO

`check_design(2)`

check_error_list

Specifies the error codes for which the **check_error** command checks.

TYPE

list

DEFAULT

CMD-004 CMD-006 CMD-007 CMD-008 CMD-009 CMD-010 CMD-011 CMD-012 CMD-014 CMD-015 CMD-016 CMD-019 CMD-026 CMD-031 CMD-037 DB-1 DCSH-11 DES-001 ACS-193 FILE-1 FILE-2 FILE-3 FILE-4 LINK-7 LINT-7 LINT-20 LNK-023 OPT-100 OPT-101 OPT-102 OPT-114 OPT-124 OPT-127 OPT-128 OPT-155 OPT-157 OPT-181 OPT-462 UI-11 UI-14 UI-15 UI-16 UI-17 UI-19 UI-20 UI-21 UI-22 UI-23 UI-40 UI-41 UID-4 UID-6 UID-7 UID-8 UID-9 UID-13 UID-14 UID-15 UID-19 UID-20 UID-25 UID-27 UID-28 UID-29 UID-30 UID-32 UID-58 UID-87 UID-103 UID-109 UID-270 UID-272 UID-403 UID-440 UID-444 UIO-2 UIO-3 UIO-4 UIO-25 UIO-65 UIO-66 UIO-75 UIO-94 UIO-95 EQN-6 EQN-11 EQN-15 EQN-16 EQN-18 EQN-20

GROUP

acs_variables

DESCRIPTION

This variable specifies the error codes for which the **check_error** command checks. The **check_error** command returns a 1 if any of the specified error codes have been generated by a previous command in the current session.

You can use this capability to stop batch jobs in which the specified error codes occur.

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

SEE ALSO

`check_error(2)`

clock_attributes

Contains attributes placed on clocks.

DESCRIPTION

Contains attributes that can be placed on clocks.

To set an attribute, use the command identified in the individual description of that attribute. To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Clock Attributes

dont_touch_network

When a design is optimized, **compile** assigns **dont_touch** attributes to all cells and nets in the transitive fanout of **dont_touch_network** ports. The **dont_touch** assignment stops at the boundary of storage elements. An element is recognized as storage only if it has setup or hold constraints. Set with **set_dont_touch_network**.

fix_hold

Specifies that **compile** should attempt to fix hold violations for timing endpoints related to this clock. Set with **set_fix_hold**.

max_fall_delay

A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_rise_delay

A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow

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 technology library. Set with **set_max_time_borrow**.

min_fall_delay

A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay

A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

period

Assigns a value to the clock period. The clock period (or cycle time) 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. Set with **create_clock -period period_value**.

hold_uncertainty

Specifies a negative uncertainty from the edges of the ideal clock waveform. Set with **set_clock_uncertainty -hold**.

setup_uncertainty

Specifies a positive uncertainty from the edges of the ideal clock waveform. Set with **set_clock_uncertainty -setup**.

propagated_clock

Specifies that the clock edge times be delayed by propagating the values through the clock network. If this attribute is not present, ideal clocking is assumed. Set with **set_propagated_clock**.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
attributes(3)
```

collection_result_display_limit

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

TYPE

integer

DEFAULT

100

DESCRIPTION

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

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

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

SEE ALSO

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

command_log_file

The **command_log_file** variable is obsolete. Use the **sh_command_log_file** variable instead.

company

Specifies the name of the company where Synopsys software is installed. The company name is displayed on the schematics.

TYPE

string

DEFAULT

""

GROUP

system_variables

DESCRIPTION

This variable specifies the name of the company where Synopsys software is installed. The company name is displayed on the schematics.

To determine the current value of this variable use the **printvar company** command. For a list of all system variables and their current values, use the **print_variable_group system** command.

compatibility_version

Sets the default behavior of the system to be the same as the Synopsys software version specified in the variable.

TYPE

string

DEFAULT

L-2016.03

GROUP

system_variables

DESCRIPTION

This variable sets the default behavior of the system to be the same as the Synopsys software version specified in the variable. This setting provides compatibility for script command files written in previous software versions. The scripts are run on the current version of the software, so results are usually better. However, the script performs the same default actions here as it did on the specified software version.

To determine the current value of this variable use the **printvar compatibility_version** command. For a list of all system variables and their current values, use the **print_variable_group system** command.

compile_clock_gating_through_hierarchy

Controls whether the **compile** or **compile_ultra** command with the **-gate_clock** option performs clock gating through hierarchy boundaries.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When this variable is set to **true**, **compile -gate_clock** and **compile_ultra -gate_clock** are allowed to use one clock gate to gate registers in different hierarchical cells. This can increase the number of clock gating opportunities and reduce the number of clock gates.

When the value is **false** (the default), the clock gating is only performed in such a way that clock gates are in the same hierarchy cell as all registers gated by them.

A clock gating cell will not be modified or removed if it or its parent hierarchical cell is marked **dont_touch** with the **set_dont_touch** command; it will not be modified by global clock gating optimization.

To determine the current value of this variable, use the **printvar compile_clock_gating_through_hierarchy** command. For a list of all compile variables and their current values, use **print_variable_group compile**.

This variable is not supported in DC Explorer.

SEE ALSO

compile_dont_use_dedicated_scanout

Controls whether optimizations use a scan cell's dedicated scan-out pin for functional connections.

TYPE

integer

DEFAULT

1

GROUP

insert_dft_variables

DESCRIPTION

Optimizations by **place_opt**, **clock_opt**, **route_opt** and **psyn_opt** do not use a scan cell's dedicated scan-out pin for functional connections. When this variable is set to 0, optimizations can use dedicated scan-out pins for functional connections.

Dedicated scan-out pins must be identified in the technology library using the **test_output_only** attribute. Contact your ASIC Vendor to ensure that dedicated scan-out pins are correctly modeled in the library that you are using.

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

compile_enable_async_mux_mapping

Controls whether the **compile** or **compile_ultra** command tries to preserve multiplexers in the fanin cone of asynchronous register pins.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

When this variable is set to **true**, the **compile** or **compile_ultra** command tries to preserve multiplexers in the fanin cone of asynchronous register pins. More specifically, if multiplexing logic (any MUX_OP or 2-input SELECT_OP) is found that is in the fanin of only asynchronous register pins (clock, and asynchronous set and reset), then this logic is mapped to a multiplexer, and a size-only attribute is set on the multiplexer. If the MUX_OP or SELECT_OP cell is specified as dont_touch, Design Compiler does not map the cell to a multiplexer. The purpose of this is to reduce the occurrence of glitches on asynchronous register pins. The default value of this variable is true.

To determine the current value of the **compile_enable_async_mux_mapping** variable, use the **printvar compile_enable_async_mux_mapping** command.

For a list of all compile variables and their current values, use **print_variable_group compile**.

SEE ALSO

`hdlin_mux_size_only(3)`

compile_enhanced_resource_sharing

Controls whether High Level synthesis should use additional algorithms to pursuit better area through increased resource sharing.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

Controls whether High Level synthesis should use additional algorithms to pursuit better area through increased resource sharing. When this variable is set to true, additional opportunities for sharing resources are evaluated with the intent of reducing area.

To determine the current value of this variable, type **printvar compile_enhanced_resource_sharing**. For a list of **hdl** variables and their current values, type **print_variable_group compile**.

compile_instance_name_prefix

Specifies the prefix used in generating cell instance names when the **compile** command is run.

TYPE

string

DEFAULT

U

GROUP

compile_variables

DESCRIPTION

This variable specifies the prefix used in generating cell instance names when running the **compile** command.

To determine the current value of this variable, use the **printvar compile_instance_name_prefix** command. For a list of compile variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_instance_name_suffix

Specifies the suffix used for generating cell instance names when the **compile** command is run.

TYPE

string

DEFAULT

""

GROUP

compile_variables

DESCRIPTION

This variable specifies the suffix used for generating cell instance names when running the **compile** command.

To determine the current value of this variable, use the **printvar compile_instance_name_suffix** command. For a list of **compile** variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_keep_original_for_external_references

Instructs the **compile** command to keep the original design when there is an external reference to the design.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

By default, the **compile** command modifies the original copy of the designs in the current design. When the **compile_keep_original_for_external_references** variable is set to true, the original design and its subdesigns are copied and preserved (before doing any modifications during compile), if there is an external reference to this design.

For example, if there is an instance of a design named *bot, U1* in the current design named *mid*, and there is an external reference from another design named *top* that is not part of current hierarchy, then *U1* will be uniquified to a new design named *bot_0* before doing any modification to the design. Therefore, when you change the current_design to *top* and perform a link, the original *bot* design will be linked into the current_hierarchy.

Usually this is needed only when you are doing a bottom compile without setting **dont_touch** attributes on all of the subdesigns, particularly with boundary optimization turned on during compile.

If there is a **dont_touch** attribute on any of the instances of the design or in the design itself, this variable does not have any effect.

SEE ALSO

compile_log_format

Controls the format of the columns to be displayed during executions of the following commands:

```
psynopt
place_opt
clock_opt
preroute_focal_opt
create_buffer_tree
place_opt_feasibility
clock_opt_feasibility
focal_opt
route_opt
```

TYPE

string

DEFAULT

%elap_time %area %wns %tns %drc %endpoint

GROUP

compile_variables

DESCRIPTION

This variable controls the format of the columns to be displayed during executions of the following commands:

```
psynopt
place_opt
clock_opt
preroute_focal_opt
create_buffer_tree
place_opt_feasibility
clock_opt_feasibility
focal_opt
route_opt
```

The default specification is shown in the DEFAULT section above and results in an output display format similar to Table 1. The headings and order of the columns displayed

correspond to the keywords specified in the syntax. For example, "%elap_time" specifies the ELAPSED TIME column, "%area" the AREA column, and so on.

Table 1
Default Compile Log Output Format

```
Elapsed              Worst Neg   Total Neg   Design
 Time       Area       Slack       Slack    Rule Cost      Endpoint
--------  ---------  ---------  ----------  ---------
-------------------------
18:00:30   1498.0      4.12        32.2          0.0   U1/U2/
CURRENT_SECS_reg[4]
18:00:30   1498.0      4.07        31.6          0.0   U1/U2/
CURRENT_SECS_reg[4]
18:00:30   1497.0      4.07        30.6          0.0   U1/U2/
CURRENT_SECS_reg[4]
18:00:31   1499.0      3.61        27.5          0.0   U1/U2/
CURRENT_SECS_reg[4]
18:00:31   1499.0      3.58        26.1          0.0   U1/U2/
CURRENT_SECS_reg[4]
```

By default, the columns in Table 1 are nine characters wide except for the ENDPOINT column, which is 25 characters. The default precision for the floating-point data types AREA, TOTAL NET SLACK, and DESIGN RULE COST is 1 digit to the right of the decimal point; for WORST NEG SLACK, 2 digits.

There are 13 possible columns that can be displayed; only 6 are displayed in the default format. You can create a customized output format by specifying any number of the available columns, with their keywords and defaults. For example, specifying "%mem" displays the MBYTES column with a width of 6 characters and a precision of 1 digit to the right of the decimal point. When you specify "%mem", the following information is displayed horizontally under these column names: COLUMN HEADER, DATA TYPE, KEYWORD, WIDTH, PRECISION, and FORMAT.

```
  MBYTES       floating point     mem         6          1           f
```

See the section "Definitions of Column Fields with Default Values" for descriptions and contents of the fields corresponding to the column headers.

Changing Default Column Parameters

You can change the default column parameters using the optional expression (w.pf,split), as follows:

compile_log_format = " %elap_time %area %wns %tns %drc %endpoint".

string

compile_log_format = "%keyword(w.pf,split)"

The quantities w, p, f, and split are defined as follows:

w

Specifies the column width. Specifying a width less than 6 defaults the width to 6. For string data types, specifying a width greater than 99 defaults the width to 99. For decimal or floating-point data types, specifying a width greater than 25 defaults to 25.

p

For floating-point numbers only. Specifies the precision in number of digits. See also the definition of f.

f

For floating-point numbers only. Specifies the precision format; values are f or g. The value of f specifies that the precision is expressed as the number of digits to the right of the decimal point. The value of g specifies that the precision is expressed as the total number of significant digits. For example, expressing the floating-point number 13.533 with a precision of 3 in the f format reports the number as 13.533 while in the g format the number is reported as 13.5.

split

By default, if the information in a given field exceeds the column width, it pushes out the next field and the next field is not printed on a new line. Specifying split overrides the default and causes the next field to begin on a new line, starting in the correct column.

1998.02 Compile Log Format

The fields for the 1998.02 version are TRIALS, AREA, DELTA DELAY, TOTAL NEGATIVE SLACK, and DESIGN RULE COST. The DELTA DELAY field is renamed MAX DELAY COST in the 1998.08 format.

To display the same log format as the 1998.02 version, set the variable as follows:

compile_log_format = ""

Definitions of Column Fields with Default Values

The fields listed under the 5 columns are defined in the following text, showing the default values. Except where noted, units are those defined by the library.

AREA

Shows the area of the design during the optimization.

```
Data type: floating point
Keyword: are
```

```
Width: 9
Precision: 1
Format: f
```

CPU SEC

Shows the process CPU time used, in seconds.

```
Data type: decimal
Keyword: cpu
Width: 7
Precision: ignored
Format: ignored
```

DELTA DELAY

See MAX DELAY COST.

DESIGN RULE COST

Measures the distance between the actual results and user-specified design rule constraints.

```
Data type: floating point
Keyword: drc
Width: 9
Precision: 1
Format: f
```

ELAPSED TIME

Tracks the elapsed time since the beginning of the current execution.

```
Data type: string
Keyword: elap_time
Width: 9
Precision: ignored
Format: ignored
```

MAX DELAY COST

Shows the current maximum delay cost of the design, which is the sum of the worst negative slack (max_path violation) at each timing endpoint.

```
Data type: floating point
Keyword: max_delay
Width: 9
Precision: 2
Format: f
```

MBYTES

Shows the process memory used, in mbytes.

```
Data type: floating point
Keyword: mem
Width: 6
Precision: 1
Format: f
```

MIN DELAY COST

Shows the current minimum delay cost of the design, which is the sum of the worst negative slack (min_path violation) at each timing endpoint.

```
Data type: floating point
Keyword: min_delay
Width: 9
Precision: 2
Format: f
```

TIME OF DAY

Shows the current time.

```
Data type: string
Keyword: time
Width: 8
Precision: ignored
Format: ignored
```

TOTAL NEG SLACK

Shows the sum of the negative slack across all endpoints in the design.

```
Data type: floating point
Keyword: tns
Width: 9
Precision: 1
Format: f
```

TRIALS

Tracks the number of transformations that the optimizer tries before making the current selection.

```
Data type: decimal
Keyword: trials
Width: 6
```

```
Precision: ignored
Format: ignored
```

WORST NEG SLACK

Shows the worst negative slack (max_path violation) in all path groups.

```
Data type: floating point
Keyword: wns
Width: 9
Precision: 2
Format: f
```

ENDPOINT

Shows the current endpoint being on which work is being done. When the delay violation is being fixed, the object for the ENDPOINT is a cell or a port. When the design rule violations are being fixed, the object for the ENDPOINT is a net.

```
Data type: string
Keyword: endpoint
Width: 25
Precision: ignored
Format: ignored
```

PATH GROUP

Shows the current path group of a valid endpoint.

```
Data type: string
Keyword: group_path
Width: 10
Precision: ignored
Format: ignored
```

DYNAMIC POWER

Shows the dynamic power of the design during optimization.

```
Data type: floating point
Keyword: dynamic_power
Width: 9
Precision: 4
Format: f
```

LEAKAGE POWER

Shows the leakage power of the design during optimization.

```
Data type: floating point
Keyword: leakage_power
Width: 9
Precision: 4
Format: f
```

TOTAL POWER

Shows the total power of the design during optimization. (total_power = dynamic_power + leakage_power)

```
Data type: floating point
Keyword: total_power
Width: 9
Precision: 4
Format: f
```

EXAMPLES

The following example increases the precision of the WORST NEG SLACK column by 1 (to 3 digits from its default of 2 digits):

```
prompt> set compile_log_format {%elap_time %area %wns(.3) %tns %drc
%endpoint}
```

The following example replaces the TOTAL NEG SLACK column in the default format with the CPU column:

```
prompt> set compile_log_format {%elap_time %area %wns %cpu %drc
%endpoint}
```

In the following example, if the ENDPOINT value exceeds the column width, the next field begins on a new line, starting in the correct column:

```
prompt> set compile_log_format {%elap_time %wns %endpoint(19,split)
%group_path}
```

The following example displays only the MIN DELAY COST column, changes the column width to 12 characters from the default of 9, and expresses the value with a precision of 3 significant digits:

```
prompt> set compile_log_format {%min_delay(12.3g)}
```

The following example sets the log format to the same format as the 1998.02 version:

```
prompt> set compile_log_format = ""
```

To determine the current value of this variable, use the **printvar compile_log_format** command. For a list of all compile variables and their current values, use **print_variable_group compile**.

SEE ALSO

```
psynopt(2)
place_opt(2)
clock_opt(2)
preroute_focal_opt(2)
create_buffer_tree(2)
place_opt_feasibility(2)
clock_opt_feasibility(2)
focal_opt(2)
route_opt(2)
```

compile_no_new_cells_at_top_level

Controls whether the **compile** command adds new cells to the top-level design.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When this variable is set to **true**, no new cells are added to the top-level design of the hierarchy during **compile**. New cells are added only to lower levels.

This variable is used when the original design has no top-level cells, to prevent the addition of new cells when adding buffers for timing optimization and design rule fixes.

If the design has leaf cells at the top level, the cells are optimized as normal, so this variable should be set to **false** (the default). Setting this variable to **true** means that **compile** will not add any cells during buffering and design rule fixing, even if the design is flat.

Note that if **set_isolate_ports** requires insertion of new cells at the top level, then these cells will be added even if **compile_no_new_cells_at_top_level** is set to **true**. This is because port isolation takes precedence.

To determine the current value of this variable, use the **printvar compile_no_new_cells_at_top_level** command. For a list of all compile variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_power_domain_boundary_optimization

Disables boundary optimization across power domain boundaries when set to false.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

This variable controls whether to allow boundary optimization across all power domain boundaries. By default, boundary optimization across power domain boundaries is enabled in **compile_ultra**, unless specifically disabled by a command such as **set_boundary_optimization**. This variable provides an automatic way to disable boundary optimization across all power domain boundaries.

SEE ALSO

compile_retime_exception_registers

Controls whether registers with common path exceptions, including max_path, min_path, multicycle_path, false_path, and group_path, can be moved by adaptive retiming.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

This variable controls whether registers with common path exceptions, including max_path, min_path, multicycle_path, false_path, and group_path, can be moved by adaptive retiming.

For example, if an SDC script contains the following command where **reg** is a register, there is a path exception on **reg**:

```
set_max_delay 1.0 -to reg
```

The variable only affects flows that use the **compile_ultra** command with the **-retime** option.

Allowed values are **false** (the default) and **true**.

If **false** is specified, adaptive retiming does not attempt to move registers with exceptions to improve timing or area.

If **true** is specified, adaptive retiming may try to move registers with the max_path, min_path, multicycle_path, false_path, and group_path path exceptions to improve timing or area.

SEE ALSO

compile_seqmap_identify_shift_registers

Controls the identification of shift registers in **compile -scan**. This feature is only supported in test-ready compile with Design Compiler Ultra with a multiplexed scan style.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

When the value of this variable is set to the default value of **true**, Design Compiler Ultra automatically identifies shift registers in the design during test-ready compile.

When all of the shift registers are identified, only the first register is mapped to a scan cell, while the remaining registers are mapped to non-scan cells. This can save a significant amount of area for designs containing many identified shift registers.

Once these shift registers are identified by Design Compiler Ultra, DFT Compiler will also recognize the identified shift registers as shift-register scan segments. But DFT Compiler will break these scan segments, if necessary, to respect test setup requirements such as maximum chain length.

Shift registers that contain synchronous logic between the registers can also be identified if the synchronous logic can be controlled such that the data can be shifted from the output of the first register to the input of the next register. This synchronous logic can either be internal to the register (for example, synchronous reset and enable) or it can be external synchronous logic (for example, multiplexor logic between the registers). For shift registers identified with synchronous logic between the registers, DFT Compiler will add additional logic to the scan-enable signal during scan insertion in order to allow the data to be shifted between the registers when in scan mode. This capability is controlled by the **compile_seqmap_identify_shift_registers_with_synchronous_logic** variable, and is enabled by default. See the

compile_seqmap_identify_shift_registers_with_synchronous_logic variable man page for details.

Shift-register identification is only supported in test-ready compile with Design Compiler Ultra with a multiplexed scan-style.

Set the **compile_seqmap_identify_shift_registers** variable to **false** if you do not want **compile_ultra -scan** to identify shift registers, or if you want to rescan the shift registers already identified in the design back to scan cells.

The **compile_seqmap_identify_shift_registers_with_synchronous_logic** variable does not have any effect when shift-register identification is disabled with the **compile_seqmap_identify_shift_registers** variable.

SEE ALSO

compile_seqmap_identify_shift_registers_with_synchronous_logic_ascii

Controls the identification of synchronous shift registers in the ASCII flow when you run the **set_scan_state test_ready** command.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When set to **true**, this variable enables the identification of synchronous shift registers for an ASCII flow when you run the **set_scan_state test_ready** command. The **compile_seqmap_identify_shift_registers** and **compile_seqmap_identify_shift_registers_with_synchronous_logic** variables must also be set to **true** to enable this capability. This flow requires a DC-Ultra license and a DFT-Compiler license.

In a binary flow, shift register identification is performed when you run the **compile_ultra -scan** command. The identified shift registers are preserved by attributes that are used by DFT Compiler to include shift registers during reporting and scan stitching.

In an ASCII flow, shift register attributes are not available. Setting this variable to **true** allows the tool to identify both regular shift registers and synchronous shift registers after reading an ASCII netlist. This identification takes place during the execution of the **set_scan_state test_ready** command.

Once the netlist is in a **test_ready** state, subsequent applications of the **set_scan_state test_ready** command do not have any effect.

If the DC-Ultra and DFT-Compiler licenses are not available, only regular shift registers are identified during the **set_scan_state test_ready** command run.

Note that the shift registers identified might be different between the binary and the ASCII flow because of the full reidentification of shift registers that takes place when this flow is enabled.

SEE ALSO

`compile_seqmap_identify_shift_registers(3)`

compile_seqmap_propagate_constants_size_only

Controls whether the **compile_ultra** command will propagate constant values through registers that are marked size_only and dont_touch when trying to identify and remove constant registers.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

When the value is **true** (the default), **compile_ultra** will propagate constant values through size_only and dont_touch registers. It will not remove the size_only and dont_touch registers, however. Even if it finds them to be constant.

When this variable is set to **false**, **compile_ultra** will assume that all size_only and dont_touch registers are nonconstant.

To determine the current value of this variable, use the **printvar compile_seqmap_propagate_constants_size_only** command. For a list of all compile variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

`set_size_only(2)`

compile_state_reachability_high_effort_merge

High effort for register merging is a new capability for the **compilecommand. It allows to find more equal and opposite registers.**

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When **compile_state_reachability_high_effort_merge** is **true**, the **compile** command tries to identify equal and opposite registers by running a more aggressive algorithm.

The **compile** command prints an OPT-1215 message whenever a register is removed from the design because it is equal or opposite to another register.

To determine the current value of this variable, use the **printvar compile_state_reachability_high_effort_merge** command. For a list of all compile variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

compile_ultra_ungroup_small_hierarchies

Determines whether to automatically ungroup small hierarchies in the **compile_ultra** flow.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

This variable, when set to **false**, turns off the automatic ungrouping of small user design hierarchies at the beginning of the **compile_ultra** flow.

Alternatively, you can achieve the same results by running the **compile_ultra** command with the **-no_autoungroup** option.

Use the **printvar compile_ultra_ungroup_small_hierarchies** command to determine the current value of this variable.

SEE ALSO

complete_mixed_mode_extraction

Enables extraction of both routed and unrouted nets with a single command. This is known as mixed-mode extraction.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

DESCRIPTION

By default, the **extract_rc** command skips unrouted or partially routed nets and extracts only completely routed nets. To estimate partially routed or unrouted nets, you must use the **extract_rc** command with the **-estimate** option. Consequently, you must run both **extract_rc** and **extract_rc -estimate** to completely extract a partially routed design.

Mixed-mode extraction simplifies this process by enabling simultaneous handling of routed and unrouted nets. By default, this variable is set to **true** enabling the following capabilities:

- The **extract_rc** command performs detailed extraction first, followed by estimation for broken or unrouted nets.
- When you run the **write_parasitics** command after doing mixed-mode extraction, it writes out mixed-mode parasitics.

To disable the mixed-mode extraction capability, set the value of this variable to **false**.

To see the current value of this variable, use the **printvar complete_mixed_mode_extraction** command.

SEE ALSO

```
extract_rc(2)
```

core_area_attributes

Contains attributes related to core area.

DESCRIPTION

Contains attributes related to core area.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class core_area -application**, the definition of attributes can be listed.

Core Area Attributes

area

Specifies area of a core area object.

The data type of **area** is double.

This attribute is read-only.

bbox

Specifies the bounding-box of a core area. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is read-only.

boundary

Specifies point list of a core area's boundary.

The data type of **boundary** is string.

This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a core area object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

direction

Specifies the direction to orient the rows in a core area.

The valid values can be horizontal and vertical.

The data type of **direction** is string.

This attribute is read-only.

is_double_back

Specifies whether a core area contains pairs of rows with one row flipped in each pair.

The data type of **is_double_back** is boolean.

This attribute is read-only.

is_flip_first_row

Specifies whether to flip the first row at the bottom of a horizontal core area or at the left of a vertical core area.

The data type of **is_flip_first_row** is boolean.

This attribute is read-only.

is_start_first_row

Specifies whether the pairing of rows starts at the bottom of a horizontal core area or at the left of a vertical core area.

The data type of **is_start_first_row** is boolean.

This attribute is read-only.

name

Specifies name of a core area object.

The data type of **name** is string.

This attribute is read-only.

num_rows

Specifies number of rows inside a core area.

The data type of **num_rows** is integer.

This attribute is read-only.

object_class

Specifies object class name of a core area, which is **core_area**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

The data type of **object_id** is integer.

This attribute is read-only.

row_density

Specifies the ratio of total height (if it's a horizontal core area) or width (otherwise) of rows to the height or width of a core area.

The data type of **row_density** is string.

This attribute is read-only.

tile_height

Specifies height of tile cell used in a core area.

The data type of **tile_height** is double.

This attribute is read-only.

tile_width

Specifies width of tile cell used in a core area.

The data type of **tile_width** is double.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

cp_full_abut_cts_region_aware

Enables clock planning and clock tree synthesis to be region or plan group aware. You must set this to true for designs with full-abut floorplan.

TYPE

binary

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

The **cp_full_abut_cts_region_aware** variable adjusts the behavior of clock planning and clock tree synthesis. Setting it to true, enables clock buffer insertion and relocation to be aware of the plan group of the location of the clock buffer and adjusts its logic hierarchy accordingly.

You must set this variable to *true* for designs with full-abut floorplan.

SEE ALSO

```
compile_fp_clock_plan(2)
cp_in_full_abut_mode(3)
```

cp_in_full_abut_mode

Enables clock planning to handle design with full-abut floorplan

TYPE

binary

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

Setting this variable to true, adjusts the clock planning behavior to generate top level clock trees that have to be distributed in different plan groups. It also implements procedures to limit the extra clock ports punched by clock planning. You need to set this variable to true for designs with full-abut floorplan.

SEE ALSO

```
compile_fp_clock_plan(2)
cp_full_abut_cts_region_aware(3)
```

cp_para_max_subjob_num

Sets the maximum number of subjobs that are created by parallel clock planning.

TYPE

integer

DEFAULT

10

GROUP

cts_variables

DESCRIPTION

Parallel clock tree planning generates clock subtrees inside plan groups using multiple parallel processes to speed up the clock planning runtime. This variable sets a limit on the number of parallel subjobs.

SEE ALSO

`compile_fp_clock_plan(2)`

cpd_skip_timing_check

Specifies whether the **check_physical_design** command performs timing checks by default.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

DESCRIPTION

This variable determines whether the **check_physical_design** command performs timing checks by default.

If this variable is **true** (the default), by default, the **check_physical_design** command skips the timing checks. This is to reduce runtime, especially for large, complex designs.

If this variable is **false**, by default, the **check_physical_design** command performs the timing checks.

The **-check_timing** option of the **check_physical_design** command overrides the default set by this variable. When you use the **-check_timing** option, the **check_physical_design** command performs the timing checks, regardless of the setting of this variable.

SEE ALSO

`check_physical_design(2)`

cto_enable_drc_fixing

This variable will be obsolete in future release.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

When the variable is set to true, DRC fixing during optimize_clock_tree will be enabled.

SEE ALSO

`optimize_clock_tree(2)`

cts_add_clock_domain_name

Specifies strings to add the clock domain name to names of cells and nets created during compile_clock_tree and balance_inter_clock_delay.

TYPE

string

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variable when set true will instruct the commands compile_clock_tree and balance_inter_clock_delay commands to add the current clock domain name to the names of buffer and nets created while synthesizing.

SEE ALSO

```
compile_clock_tree(2)
balance_inter_clock_delay(2)
```

cts_blockage_aware

Controls whether the **compile_clock_tree** command uses the blockage-aware clock tree synthesis algorithm.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

Setting this variable to false turns off the blockage-aware clock tree synthesis algorithm of the **compile_clock_tree** command.

SEE ALSO

`compile_clock_tree(2)`

cts_clock_opt_batch_mode

Runs the **compile_clock_tree**, **optimize_clock_tree**, and **balance_inter_clock_delay** commands in clock tree synthesis (CTS) batch mode when this variable is set to true.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variable runs the **compile_clock_tree**, **optimize_clock_tree**, and **balance_inter_clock_delay** in CTS batch mode when this variable is set to true.

For more information on batch mode, see the man page for the **set_cts_batch_mode** command.

SEE ALSO

```
clock_opt(2)
reset_cts_batch_mode(2)
report_cts_batch_mode(2)
set_cts_batch_mode(2)
```

cts_clock_source_is_exclude_pin

This variable controls the cascaded create-clock behavior. If this variable is set to true, clock tree synthesis (CTS) marks the clock source of a downstream **create_clock** command as an implicit exclude pin.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

This variable controls the cascaded create-clock behavior. When set to true, it enables CTS to mark the sources of cascaded create_clocks downstream as implicit exclude pins for the purpose of upstream clocks. This behavior is consistent with the timer behavior for cascaded clocks. For example, the following command:

```
prompt> create_clock CLK1
```

has a **create_clock CLK2** defined in its fanouts at pin PIN2. CTS marks PIN2 as an implicit exclude pin for CLK1. PIN2 is still a valid driver for CLK2. This de-couples the two clocks and prevents quality of results (QoR) degradation resulting from the synthesis order of CLK1 and CLK2. If CLK1 is synthesized first, PIN2 still has a huge design rule checking (DRC) violation because CLK2 has not yet been touched by CTS. This affects the results for CLK1 when CTS tries to balance an unsythesized tree with other synthesized trees.

SEE ALSO

```
compile_clock_tree(2)
create_clock(2)
```

cts_do_characterization

Specifies for clock tree synthesis to print additional information to the log file when the **cts_do_characterization** variable is set to true. It prints detailed characterization data for the buffers and inverters that clock tree synthesis uses.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variable, if set to true, prints information about buffer and inverter characterization. It prints estimated skew, target delay, driving resistance, and input capacitance for each buffer and inverter in the list. The buffer list includes all the available buffers and inverters in the library. If you use the **set_clock_tree_references** command, it provides a list that overwrites the **cts_do_characterization** list.

SEE ALSO

`set_clock_tree_references(2)`

cts_enable_clock_at_hierarchical_pin

Specifies that clock tree construction be performed bottom-up for clocks at hierarchical pins. It is for designs with clock sources or clock exceptions defined at hierarchical pins.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

Clock tree synthesis (CTS) requires that clock sources must be at points that have physical information. In general, a hierarchical pin does not have physical information for accurately calculating clock delay. Prior to version B2008.09, create_clock at hierarchical pins is excluded from CTS.

You might have to modify SDC constraint files and move the clocks to leaf cell pins. The **cts_enable_clock_at_hierarchical_pin** variable (switch) is created to minimize your manual work. By using the switch, clock tree construction is performed bottom-up for clocks at hierarchical pins. The clock source is assumed at a pin of the top-most buffer inserted by the tool.

SEE ALSO

`compile_clock_tree(2)`

cts_enable_drc_fixing_on_data

Controls whether clock tree synthesis fixes DRC violations beyond exceptions on pins reached by the clock signal, regardless of whether the pins have the **pin_on_clock_network** attribute.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variables specifies that clock tree synthesis should fix DRC violations on clock exception subtrees reached by the clock signal. The pins on these subtrees might have the **pin_on_clock_network** attribute. When this variable is enabled, clock tree synthesis might fix DRC violations that are actually on data paths. The **cts_fixed** and **cts_synthesized** attributes are set on the buffers and nets inserted by clock tree synthesis and thus cannot be modified during optimization. The net type on a data net inserted by clock tree synthesis is marked as a clock net. You can use the **-drc_violators** or **-all_drc_violators** option with the **report_clock_tree** command to report the violations on these nets that are on data paths but reachable by clocks. The **remove_clock_tree** and **mark_clock_tree** commands are also supported.

SEE ALSO

```
clock_opt(2)
compile_clock_tree(2)
```

cts_enable_priority_driven

Specifies that the clock tree synthesis tools (compile_clock_tree, optimize_clock_tree and balance_inter_clock_delay) should be run in priority_driven sequential multi-mode flow, when set to true.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variable set the clock tree synthesis tool into priority_driven sequential multi-mode flow. In priority_driven sequential multi-mode flow, user can specify with -clock trees for compile_clock_tree/ optimize_clock_tree to only work on a few clocks at a time, and use mark_clock_tree -seq_freeze to freeze the portion of the clock trees that just finished compile_clock_tree/optimize_clock_tree. The later compile_clock_tree/optimize_clock_tree including balance_inter_clock_delay will honor the markings of the seq_freeze to make sure the previously worked portion of the clock tree will not be touched again. This is beneficial when user has clear priorities of the clock to be synthesized, and less important clocks will not hurt the qor of the more important clocks.

SEE ALSO

```
compile_clock_tree(2)
optimize_clock_tree(2)
balance_inter_clock_delay(2)
mark_clock_tree(2)
```

cts_enable_rc_constraints

When this variable is set to true, CTS will apply internally derived RC constraints during clustering to reduce skew caused by wire delay. This feature is to be used with config file only.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

For a high performance design that needs to operate under very different operating conditions (PVT, TLU+), it is important for its clock tree to maintain good skew across different corners. Skew varies from corner to corner due to both cell delay and interconnect delay variations. The config file flow in ICC CTS provides a way to build a clock tree with pre-defined structure to control cell delay variation. The variable, cts_enable_rc_constraints, allows more control over interconnect delay variation by enforcing RC constraints during clustering stage.

Use the following command to determine the current value of the variable:

```
prompt> printvar cts_enable_rc_constraints
```

SEE ALSO

```
compile_clock_tree(2)
cts_rc_relax_factor(3)
```

cts_fix_clock_tree_sinks

Specifies for clock tree synthesis (CTS) to put the **cts_fixed** attribute on all sinks for optimization and legalization to honor, when set to true.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

Specifies for clock tree synthesis (CTS) to put the **cts_fixed** attribute on all sinks, except those that have already been marked fixed before CTS. Cells that have the **cts_fixed** attribute cannot be moved or sized by legalization or placement and timing optimization

By default, CTS sets the **cts_in_place_size_only** attribute on all sinks, except those that have been marked as fixed before CTS. Cells that have the **cts_in_place_size_only** attribute can be moved by the legalizer, and can be sized during placement and timing optimization.

SEE ALSO

```
compile_clock_tree(2)
set_clock_tree_options(2)
```

cts_fix_drc_beyond_exceptions

Specifies for clock tree synthesis (CTS) to fix all design rule checking (DRC) violations before skew minimization, when set to true. From the A2007.12 release forward, this tool fixes all DRC violations, including those beyond clock exceptions by default. This variable provides you with a switch to prevent DRC fixing beyond clock exceptions.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

Prior to the A2007.12 release, DRC violations beyond clock exceptions was performed by physopt. That is, any net that is not processed during CTS will be processed by physopt after CTS. However, from the 2007.12 release forward, physopt does not touch any clock nets, including those beyond clock exceptions. Clock nets are identified by timer. With this change, DRC violations beyond clock exceptions must be fixed by CTS itself. If you do not want to have CTS fix those DRC violations, you can set the switch to false.

SEE ALSO

`compile_clock_tree(2)`

cts_force_user_constraints

Specifies for clock tree synthesis (CTS) to ignore library constraints for capacitance and transition, when set to true. This variable is useful for comparing CTS results between IC Compiler and other tools.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variable enables the **compile_clock_tree** command to ignore the transition and capacitance constraints from the library when it is set to true. User constraints or default CTS constraints prevail. In addition to constraints from the library, the SDC constraints specified will also get ignored when this variable is set to true. This is useful while comparing results with CTS in other tools.

SEE ALSO

`compile_clock_tree(2)`

cts_honor_detail_routed_nets

When this variable is set to true, clock_opt, compile_clock_tree, optimize_clock_tree and balance_inter_clock_delay will not do global routing of pre-routed clock nets and will use the existing detail routes for more accurate delay computation.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

When this variable is set to true, clock_opt, compile_clock_tree, optimize_clock_tree and balance_inter_clock_delay will honor existing detail routes on clock nets. The commands will skip rerouting these pre-routed clock nets and will use the existing detail routes for more accurate delay estimation. Global routed clock nets might still get rerouted even when this variable is true.

SEE ALSO

```
clock_opt(2)
compile_clock_tree(2)
balance_inter_clock_delay(2)
```

cts_icgr_enforce_voltage_areas

Setting this variable to false will allow integrated clock global router (ICGR) to route over the voltage areas.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

This variable controls whether ICGR routes over the voltage areas. By default, ICGR will not route over the voltage areas. If it is required that ICGR should route over the voltage areas, set this variable to false.

SEE ALSO

`optimize_clock_tree(2)`

cts_instance_name_prefix

Specifies string to prepend to names of cells created during compile_clock_tree and optimize_clock_tree.

TYPE

string

DEFAULT

""

GROUP

cts_variables

DESCRIPTION

This variable specifies a string to prepend to the names of buffers and inverters created during compile_clock_tree and optimize_clock_tree. It provides for a convenient means of collecting all new cells resulting from a given run of the compile_clock_tree or optimize_clock_tree command.

SEE ALSO

```
compile_clock_tree(2)
optimize_clock_tree(2)
cts_net_name_prefix(3)
```

cts_mesh_net_name

Specifies that clock tree synthesis (CTS) is to treat the net specified as ideal and skip it for synthesis and optimization. This variable is useful for processing the sub-trees below tap drivers in a multisource CTS flow.

TYPE

string

DEFAULT

""

GROUP

cts_variables

DESCRIPTION

This variable informs the compile_clock_tree and optimize_clock_tree commands to ignore the delay of this net driving tap_drivers by treating it as an ideal net. Compile_clock_tree will not attempt to balance latencies among the sub-trees, but optimize_clock_tree will insert or adjust delays as necessary to balance the sub-trees. This is useful when running with the split_multisource_clock command: it obviates the need to define temporary clocks on each of the tap_drivers and to balance them with the balance_interclock_delay command.

SEE ALSO

`split_multisource_clock(2)`

cts_move_clock_gate

This variable controls the initial relocation of existent clock gates in compile_clock_tree.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

When this variable is set to true, compile_clock_tree will relocate existent clock gates to their fanout centers before synthesis in order to improve insertion delay and clock skew.

SEE ALSO

`compile_clock_tree(2)`

cts_mv_check_bufferable_hier_nets

Enables checking over hierarchical clock net segments for selecting bufferable net segment during the multivoltage clock tree synthesis flow before skipping the net with CTS-615.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variable, when set to true, enables checking over all the hierarchical clock net segments to select bufferable net segment during the multivoltage clock tree synthesis flow before skipping the net with CTS-615. With this variable false, only the clock rootnet (hierarchical parent net driving all loads) is checked for bufferability and if this net has multivoltage dont touch then the net is skipped with CTS-615.

SEE ALSO

```
compile_clock_tree(2)
clock_opt(2)
CTS-615(n)
```

cts_mv_dummy_hierarchy

Enables the creation of dummy hierarchies during clock tree synthesis and optimization for multivoltage designs.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

This variable, if set to true, enables the creation of dummy hierarchies by the **compile_clock_tree**, **optimize_clock_tree, and clock_optcommands. They are created to minimize port punching when buffering has to happen in a power domain which is different from the power domains of the driver and the loads of the net being buffered. This variable is only active if the cts_mv_enhanced_robustness variable is set to true.**

SEE ALSO

```
compile_clock_tree(2)
optimize_clock_tree(2)
clock_opt(2)
cts_mv_enhanced_robustness(3)
```

cts_mv_enhanced_robustness

Enables the improvements to the robustness of the multivoltage clock tree synthesis and optimization flow.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

This variable, when set to true, enables enhancement to the robustness of the multivoltage clock tree synthesis and optimization flow.

SEE ALSO

```
compile_clock_tree(2)
optimize_clock_tree(2)
clock_opt(2)
```

cts_net_name_prefix

Specifies string to prepend to names of nets created during compile_clock_tree and optimize_clock_tree.

TYPE

string

DEFAULT

""

GROUP

cts_variables

DESCRIPTION

This variable specifies a string to prepend to the names of nets created during compile_clock_tree and optimize_clock_tree. It provides for a convenient means of collecting all new nets resulting from a given run of the compile_clock_tree or optimize_clock_tree command.

SEE ALSO

```
compile_clock_tree(2)
optimize_clock_tree(2)
cts_instance_name_prefix(3)
```

cts_new_one_fanout_path

Controls whether clock tree synthesis inserts delay cells in a detour pattern when a significant amount of clock delay is required.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

When clock tree synthesis work to optimize skew between different branches of a clock or optimizes interclock skew between different clocks, it might need to add delay to certain clock paths. This variable specifies that when the additional delay is significant and needs to be achieved using multiple stages of delay cells, clock tree synthesis should insert delay cells with space between each stage and form a detouring chain of cells. As a result net delay is also be used to achieve the delay target.

This feature results in a more efficient use of delay cells to achieve the target delay. Also, it strikes a better balance of cell delay and net delay on the clock path, which is beneficiary for multicorner clock skew.

SEE ALSO

```
clock_opt(2)
optimize_clock_tree(2)
balance_inter_clock_delay(2)
```

cts_ocv_path_sharing_derating_threshold

Specifies the threshold for the maximum timing derate range beyond which OCV path sharing is enabled during the clock tree synthesis.

TYPE

float

DEFAULT

0.09

GROUP

cts_variables

DESCRIPTION

The maximum timing derate range is the largest deference between the early and late timing derates of any clock net or cell in all cts-mode scenarios of the design. If the maximum timing derate range exceeds the threshold specified by using this variable, the tool synthesizes the clock trees in the OCV path sharing mode. If the maximum timing derate range is smaller than the threshold specified by using this variable, the tool disables the OCV path sharing mode. When the timing derates ranges are small the benefits of enabling the OCV path sharing mode is negligible.

SEE ALSO

```
compile_clock_tree(2)
set_clock_tree_options(2)
```

cts_prects_upsize_gates

Specifies for the **compile_clock_tree** command to up-size the original clock gates to the LEQ cell with the highest driving strength, when set to true.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

This variable specifies for the **compile_clock_tree** command to up-size the original clock gates to the LEQ cell with the highest driving strength. Up-sizing sizes all the original gates present in a clock tree to the highest available driving LEQ cell. Gates marked as fixed or dont size are not sized. Currently, this variable also sizes integrated clock gatings (ICGs).

In some cases LEQ cells might not be available in the library. In other cases, LEQ cells might be marked as dont use. In the later case, you are advised to remove the **dont_use** attribute from library cells.

This variable controls only the up-sizing during the preprocessing step of the **compile_clock_tree** command. This is different from the gate-sizing optimization step during Embedded-CTO and Standalone-CTO.

SEE ALSO

`compile_clock_tree(2)`

cts_priority_driven_redo_drc_tree_for_normal

Specifies that during sequential multi-mode CTS flow, the drc tree portion seen as normal for the current clock list should be redo as normal tree.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

Due to the existance of overlapping among clock trees, a portion of the clock tree can be normal (that need skew/delay balance) for some clock while being beyond CTS exception (that does not need skew/delay balance) for some other clocks. With CTS sequential multi-mode, the clocks that are exception for the overlapped portion can be built first. This overlapped portion will not have skew/delay balanced but only drc fixed, which can be bad for skew of the normal clock.

This varaible, when set to true, will remove this overlapped portion if the above situation happens and rebuild this portion as normal tree with delay balancing. User is responsible to make sure that the constraint of the first clock (that is exception for the overlappend portion) is not tighter than second clock (that is normal for the overlappend portion), in order to avoid DRC violations.

SEE ALSO

```
compile_clock_tree(2)
optimize_clock_tree(2)
mark_clock_tree(2)
cts_enable_priority_driven(3)
```

cts_process_net_with_mv_violation

Controls if the compile_clock_tree command processes nets with multivoltage violations.

TYPE

string

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variable, when set true, instructs the compile_clock_tree command to process nets that have multivoltage violations. If set to false, the default, the compile_clock_tree skips these nets.

SEE ALSO

`compile_clock_tree(2)`

cts_push_down_buffer

Enables the **split_clock_net** command to push down clock sinks by creating new cell instances to drive them, when set to true.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variable specifies that if one net or a list of nets is specified as the object list by the **split_clock_net** command and part or all of fanout pins of those nets are clock sinks, these clock sinks are to be pushed down one gate level with a new cell instance of a certain buffer type to drive them. The new cell instance can then be cloned to several cell instances as other integrated clock gating (ICG) gates are split.

SEE ALSO

`split_clock_net(2)`

cts_rc_relax_factor

This variable controls the internally derived RC constraints for clustering.

TYPE

float

DEFAULT

1

GROUP

cts_variables

DESCRIPTION

Tighter RC constraints will lead to smaller clusters and more resources. This variable allows trade-offs between resource and skew variation from wire delay. Setting the value larger than one results in more relaxed RC constraints and less resource requirement.

Use the following command to determine the current value of the variable:

```
prompt> printvar cts_rc_relax_factor
```

SEE ALSO

```
compile_clock_tree(2)
cts_enable_rc_constraints(3)
```

cts_region_aware

Setting this variable to false will turn off the region aware clock tree synthesis algorithm in compile_clock_tree.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

Setting this variable to false will turn off the region aware clock tree synthesis algorithm in compile_clock_tree.

SEE ALSO

`compile_clock_tree(2)`

cts_self_gating_connect_scan_enable

With this variable set to true, self gating ICG (integrated clock gate) scan enable pin will be connected to one of the driven registers scan enable net.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

With this variable set to true, self gating ICG (integrated clock gate) scan enable pin will be connected to one of the driven registers scan enable net. By default, or setting the variable to false, the scan enable pin on inserted ICG will be left open. Command insert_self_gating_dft_logic has an option icg_test_enable_driver is for the user to control the ICG scan enable.

Use the following command to determine the current value of the variable:

```
prompt> printvar cts_self_gating_connect_scan_enable
```

SEE ALSO

```
compile_clock_tree(2)
insert_self_gating_dft_logic(2)
```

cts_self_gating_data_toggle_rate_filter

This variable is for user to set a limit value on register data/control pin toggle rate for register selection during self gating.

TYPE

float

DEFAULT

0.001

GROUP

cts_variables

DESCRIPTION

This variable is for user to set a limit value on register data/control pin toggle rate for register selection during self gating. In the register selection, the value is normalized with the minimum clock period.

Use the following command to determine the current value of the variable:

```
prompt> printvar cts_self_gating_data_toggle_rate_filter
```

SEE ALSO

`compile_clock_tree(2)`

cts_self_gating_num_regs

This variable is for user to set a maximum number of registers that can be self gated.

TYPE

integer

DEFAULT

20000

GROUP

cts_variables

DESCRIPTION

This variable is for user to set a maximum number of registers that can be self gated. This is to avoid over congest the design and affect timing down the flow.

Use the following command to determine the current value of the variable:

```
prompt> printvar cts_self_gating_num_regs
```

SEE ALSO

`compile_clock_tree(2)`

cts_self_gating_num_regs_max_percent

This variable is for user to set a maximum percent of registers that can be self gated.

TYPE

integer

DEFAULT

24

GROUP

cts_variables

DESCRIPTION

This variable is for user to set a maximum percent of registers that can be self gated. This is to avoid over congest the design and affect timing down the flow.

Use the following command to determine the current value of the variable:

```
prompt> printvar cts_self_gating_num_regs_max_percent
```

SEE ALSO

```
compile_clock_tree(2)
cts_self_gating_num_regs(3)
cts_self_gating_num_regs_max_percent_total_cell(3)
```

cts_self_gating_num_regs_max_percent_total_cell

This variable is for user to set a maximum percent of all cells in the design that can be self gated.

TYPE

integer

DEFAULT

7

GROUP

cts_variables

DESCRIPTION

This variable is for user to set a maximum percent of all cells in the design that can be self gated. This is to avoid over congest the design and affect timing down the flow.

Use the following command to determine the current value of the variable:

```
prompt> printvar cts_self_gating_num_regs_max_percent_total_cell
```

SEE ALSO

```
compile_clock_tree(2)
cts_self_gating_num_regs(3)
cts_self_gating_num_regs_max_percent(3)
```

cts_self_gating_reg_power_filter

This variable is for user to set a limit value on register power for register selection during self gating.

TYPE

float

DEFAULT

0.001

GROUP

cts_variables

DESCRIPTION

This variable is for user to set a limit value on register power for register selection during self gating. Any register power less than this value is skipped from self gating.

Use the following command to determine the current value of the variable:

```
prompt> printvar cts_self_gating_reg_power_filter
```

SEE ALSO

`compile_clock_tree(2)`

cts_self_gating_wns_backoff

This variable is for user to set a value on register data pin timing slack for register selection to avoid timing degrade from self gating.

TYPE

float

DEFAULT

0.23

GROUP

cts_variables

DESCRIPTION

This variable is for user to set a value on register data pin timing slack for register selection to avoid timing degrade from self gating. The back off value is the minimum clock period multiply by the variable value. If a register's data pin timing slack is within the path group WNS and the back off value, the register is skipped from self gating.

Use the following command to determine the current value of the variable:

```
prompt> printvar cts_self_gating_wns_backoff
```

SEE ALSO

`compile_clock_tree(2)`

cts_target_cap

Sets target capacitance value for clock tree synthesis (CTS) clustering instead of using the tool-calculated target based on maximum capacitance.

TYPE

float

DEFAULT

0

GROUP

cts_variables

DESCRIPTION

This variable specifies that when constraint capacitance is less than target capacitance, the tool continues to use the constraint for design rule checking (DRC) fixes; however, the tool uses the user specified target for clustering. Normally, user input target capacitance overrides internal target capacitance in the tool.

Explore all other known possibilities of quality of results (QoR) improvement before changing the CTS target. Changing the target impacts clustering and CTS QoR. Making the target too tight (< 0.2), often degrades results. No easy way exists to know the best target for a design.

Change targets only when constraints are greater than the target. Do not change the targets for regular scenarios. Changing target capacitance for regular scenarios is not a recommended usage.

SEE ALSO

```
compile_clock_tree(2)
set_clock_tree_options(2)
```

cts_target_transition

Sets target transition value for clock tree synthesis (CTS) clustering instead of using the tool-calculated target based on maximum transition.

TYPE

float

DEFAULT

0

GROUP

cts_variables

DESCRIPTION

This variable specifies that when constraint transition is less than target transition, the tool continues to use the constraint for design rule checking (DRC) fixes; however, the tool uses the user specified target for clustering. Normally, user input target transition always overrides internal target transition in the tool.

Explore all other known possibilities of quality of results (QoR) improvement before changing the CTS target. Changing target impacts clustering and CTS QoR. Making the target too tight (< 0.2), often degrades results. No easy way exists to know the best target for a design.

Change targets only when constraints are greater than the target. Do not change the targets for regular scenarios. Changing target transition for regular scenarios is not a recommended usage.

SEE ALSO

```
compile_clock_tree(2)
set_clock_tree_options(2)
```

cts_traverse_dont_touch_subtrees

Specifies for **compile_clock_tree** command to set clock net type and routing properties for nets beyond dont_touch_subtree exceptions, when set to true.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

When this variable is set to true (the default value), the **compile_clock_tree** command updates the net attributes for nets beyond the exceptions specified by the **dont_touch_subtree** attribute. Routing rule and layer list will not be affected.

SEE ALSO

```
compile_clock_tree(2)
set_clock_tree_exceptions(2)
set_clock_tree_options(2)
```

cts_use_arnoldi_based_delays

Enables the use of arnoldi delays during clustering in clock tree synthesis.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variable specifies that clock tree synthesis should use arnoldi delay model while doing clustering. By default clock tree synthesis uses elmore delay model. Using arnoldi delay model during clustering in clock tree synthesis results in matching the delay model used during optimization steps after clock tree synthesis.

Arnoldi delay model requires nets to be global routed. Turning on this variable enables integrated clock global routing during clustering in clock tree synthesis.

SEE ALSO

```
clock_opt(2)
compile_clock_tree(2)
```

cts_use_debug_mode

Specifies for clock tree synthesis (CTS) to print additional debugging information in the log file, when set to true.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

This variables specifies for CTS to print additional information, including data such as the user-defined design rule constraints (maximum transition time, maximum capacitance, and maximum fanout); the target design rule constraints set by the tool (maximum transition time, maximum capacitance, and maximum fanout); and the user-defined clock tree timing constraints (skew, insertion delay, and levels per net).

SEE ALSO

`compile_clock_tree(2)`

cts_use_lib_max_fanout

When set to true, clock tree synthesis will honor max fanout constraint set on library cell pins.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

Max fanout constraints set on library cell pins are usually for logic synthesis purpose. Setting cts_use_lib_max_fanout to true will allow CTS to honor library max fanout constraints.

SEE ALSO

`compile_clock_tree(2)`

cts_use_sdc_max_fanout

When set to true, clock tree synthesis will honor SDC max fanout constraint set on current design.

TYPE

Boolean

DEFAULT

false

GROUP

cts_variables

DESCRIPTION

SDC max fanout constraint set on current design is usually for logic synthesis purpose. Setting cts_use_sdc_max_fanout to true will allow CTS to honor SDC max fanout constraint set on current design. Note that CTS does not support SDC max fanout constraints set on other objects.

SEE ALSO

`compile_clock_tree(2)`

db_load_ccs_data

This variable is obsolete. The tool automatically loads the CCS timing information.

dct_placement_ignore_scan

Sets the flag for the placer to ignore scan connections in Design Compiler topographical.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

Setting the **dct_placement_ignore_scan** variable to **true** flags the placer to ignore scan connections, particularly for a scan-stitched design. This is similar to the **create_placement** command with the **-ignore_scan** option in IC Compiler.

SEE ALSO

ddc_allow_unknown_packed_commands

Causes the **read_file** command to attempt to read .ddc files that contain packed commands that are unknown to the current version of **dc_shell**.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** (the default), the **read_file** command attempts to read a .ddc file that contains embedded commands (constraints) that are unknown to the current version of **dc_shell**. This might allow the tool to read a .ddc file that was written by a newer version of **dc_shell** which, for some reason, is not currently available.

If this feature is enabled, any unrecognized commands are discarded, possibly resulting in the loss of important data. It is best that .ddc files be read by the same (or later) version of the tool as was used to write them out.

If the variable is set to **false**, any attempt to read a file with unknown embedded commands results in a read failure and DDC-6 error message. You might want to set this variable to **false** as a check against possible loss of constraint data.

It may not be possible to read certain DDC files written by newer versions of the tool, even with this variable set to **true**.

This variable is only evaluated during the **read_file** command. Unknown commands read in from DDC files while this variable is **true** will continue to be skipped, even if the variable is subsequently set to **false**.

SEE ALSO

```
read_file(2)
DDC-6(n)
DDC-12(n)
DDC-13(n)
```

def_enable_ndr_softspacing

Enables the **write_def** and **read_def** commands to process soft spacing weights of nondefault routing rules in the database.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable determines whether the **read_def** and **write_def** commands consider the soft spacing weights of nondefault routing rules in the database. You can specify spacing weights between specific layers by using the **define_routing_rule** command with the **-spacing_weights** option.

By default, the variable is set to **false** and the **read_def** and **write_def** commands do not consider spacing weights. The **read_def** command implements each spacing value as hard rule. If the nondefault routing rule is soft (spacing weight less than 1.0), the **write_def** command ignores it and writes out the minimum spacing defined in the technology file. The HARDSPACING keyword is not written to the DEF file.

When this variable is set to **true**, the **read_def** and **write_def** commands consider spacing weights. If a soft rule (spacing weight less than 1.0) is specified for a layer, the **read_def** command implements the nondefault spacing rule as a soft rule with the weight set to 0.5. The **write_def** command writes out any weighted spacing value to the DEF file. If the nondefault spacing rule is a hard rule (spacing weight of 1.0), the spacing in the nondefault rule is written with the HARDSPACING keyword. Due to the limitations of the DEF syntax, the weight value cannot be described in the DEF file.

SEE ALSO

```
write_def(2)
read_def(2)
define_routing_rule(2)
report_routing_rules(2)
```

def_enable_no_legalize_name

Controls whether the **read_def** command uses alternate name styles to solve name mismatch issues.

TYPE

Boolean

DEFAULT

false

GROUP

read_def

DESCRIPTION

By default, if the cell or net name specified in DEF does not properly match the existing design object, the **read_def** creates a new cell or net. When you set this variable to **true** before using the **read_def** command, it tries to solve the name mismatch, but it might require more runtime.

When the command finds an unmatched cell or net, it tries alternate names that contain name escaping to try to match to the existing design names. Note that in the case of several design objects with similar, escaped names it is possible that a match might be made to a design object that you did not intend, so use caution when enabling this variable.

For example, assume that a cell instance name in the design is A\/B and the DEF file defines a COMPONENT named A/B. When this variable is enabled, the **read_def** command can match the component named A/B to the instance named A\/B.

To determine the current value of this variable, enter **printvar def_enable_no_legalize_name**.

SEE ALSO

```
read_def(2)
write_def(2)
```

def_non_default_width_wiring_to_net

Controls whether the **write_def** command writes nets with nondefault width to the DEF NETS section or SPECIALNETS section.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Controls whether the **write_def** command writes nets with nondefault width to the DEF NETS section or SPECIALNETS section.

By default (**false**), the **write_def** command writes nets with nondefault width to the SPECIALNETS section.

If you set the variable to **true**, the **write_def** command creates a nondefault routing rule named SNPS_DefGenNDR_<n> for each nondefault width and writes the nets with nondefault width to the DEF NETS section with a TAPERRULE statement, which makes the output DEF file less dependent on the user-specified nondefault routing rules.

The **read_def** command creates a nondefault routing rule in the design cell if it finds a definition in the NONDEFAULTRULES section, unless its prefix is SNPS_DefGenNDR_<n>.

To determine the current value of this variable, enter **printvar def_non_default_width_wiring_to_net**.

SEE ALSO

```
read_def(2)
write_def(2)
define_routing_rule(2)
```

default_input_delay

Specifies the global default input delay value to be used for environment propagation.

TYPE

float

DEFAULT

30

GROUP

acs_variables

DESCRIPTION

Specifies the global default input delay value to be used for environment propagation. This variable is used by the **derive_constraints** command.

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

SEE ALSO

`derive_constraints(2)`

default_name_rules

Contains the name of a name rule to be used as a default by the **change_names** command, if the command's **-rules** option does not specify a *name_rules* value.

TYPE

string

DEFAULT

""

GROUP

system_variables

DESCRIPTION

This variable contains the name of a name rule to be used as a default by the **change_names** command, if the command's **-rules** option does not specify a *name_rules* value.

The **change_names** command changes the names of ports, cells, and nets to conform to the rules specified by **default_name_rules**. The *name_rule* you assign to **default_name_rules** must already have been defined using **define_name_rules**. For information on the format and creation of *name_rules*, see the **define_name_rules** man page.

To determine the current value of this variable, use the **printvar default_name_rules** command. For a list of all system variables and their current values, use **print_variable_group system**.

SEE ALSO

```
change_names(2)
define_name_rules(2)
system_variables(3)
```

default_output_delay

Specifies the global default output delay value to be used for environment propagation.

TYPE

float

DEFAULT

30

GROUP

acs_variables

DESCRIPTION

This variable specifies the global default output delay value to be used for environment propagation. This variable is used by the **derive_constraints** command.

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

SEE ALSO

`derive_constraints(2)`

default_port_connection_class

Contains the value of the connection class to be assigned to ports that do not have a connection class assigned to them.

TYPE

string

DEFAULT

universal

GROUP

compile_variables

DESCRIPTION

This variable specifies the value of the connection class to be assigned to ports that do not have a connection class assigned to them. The default value for **default_port_connection_class** is **universal**.

To determine the current value of this variable, use the **printvar default_port_connection_class** command. For a list of all compile variables and their current values, use the **print_variable_group compile** command.

SEE ALSO

`set_connection_class(2)`

default_schematic_options

Specifies options to use when schematics are generated.

TYPE

string

DEFAULT

-size infinite

GROUP

```
schematic_variables
view_variables
```

DESCRIPTION

This variable options to use when schematics are generated. When the value is set to **-size infinite** (the default), the schematic for a design is displayed on a single page. This is used by the Design Analyzer.

To determine the current value of this variable, use the **printvar default_schematic_options** command. For a list of all schematic or view variables and their current values, use either the **print_variable_group schematic** or the **print_variable_group view** command.

derive_pg_preserve_floating_tieoff

Preserves floating tie-off when executing the **derive_pg_connection** command.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

This variable controls the behavior of the **derive_pg_connection [-tie | -all]** command with respect to the hierarchical tie-off pins that have no leaf-level loads.

By default, the **derive_pg_connection** command keeps the floating hierarchical tie-off pins that have no leaf-level loads connected to tie-off nets. Thus, they are not connected to power or ground nets. As a result, the Verilog output has 1’b1 or 1’b0 for these floating hierarchical tie-off pins. For the tool to derive power or ground nets for floating tie-off pins, set this variable to **false** before using the **derive_pg_connection** command.

SEE ALSO

`derive_pg_connection(2)`

design_attributes

DESCRIPTION

Contains attributes that can be placed on a design.

There are a number of commands used to set attributes. Most attributes, however, can be set with the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute. Otherwise, use **set_attribute**. If an attribute is "read only," it cannot be set by the user.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Design Attributes

async_set_reset_q

Establishes the value (0 or 1) that should be assigned to the q output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_qn**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then, by default, if set and reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.

Note: If you are unsure whether your technology library uses V3.0a syntax, ask your ASIC vendor.

async_set_reset_qn

Establishes the value (0 or 1) that should be assigned to the qn output of an inferred register if set and reset are both active at the same time. To be used with **async_set_reset_q**. Use these attributes only if you have used the **one_hot** or **one_cold** attributes/directives in your HDL description *and* your technology library is written using pre-V3.0a syntax; *or* if your technology library does not use a consistent convention for q and qn when set and reset are both active. If a V3.0a or later syntax technology library is used, then, by default, if set and

reset are both active at the same time Design Compiler will use the convention of the selected technology library (**target_library**). Set with **set_attribute**.

Note: If you are unsure whether your technology library uses V3.0a syntax, ask your ASIC vendor.

balance_registers

Determines whether the registers in a design are retimed during **compile**. When **true** (the default value), **compile** invokes the **balance_registers** command, which moves registers to minimize the maximum register-to-register delay. Set this attribute to **false**, or remove it, to disable this behavior.

Set with **set_balance_registers**.

If your design contains generic logic, you should ensure that all components are mapped to cells from the library before setting the **balance_registers** attribute.

boundary_optimization

Enables **compile** to optimize across hierarchical boundaries. Hierarchy is ignored during optimization for designs with this attribute set to **true**. Set with **set_boundary_optimization**.

default_flip_flop_type

Specifies the default flip-flop type for the current design. During the mapping process, **compile** tries to convert all unmapped flip-flops to this type. If **compile** is unable to use this flip-flop, it maps these cells into the smallest flip-flop possible. Set with **set_register_type -flip_flop flip_flop_name**.

default_flip_flop_type_exact

Specifies the exact default flip-flop type for the current design. During the mapping process, **compile** converts unmapped flip-flops to the exact flip-flop type specified here. Set with **set_register_type -exact -flip_flop flip_flop_name**.

default_latch_type_exact

Specifies the exact default latch type for the current design. During the mapping process, **compile** converts unmapped latches to the exact latch type specified here. Set with **set_register_type -exact -latch latch_name**.

design_type

Specifies the type of the current design. Allowed values are **fsm** (finite state machine); **pla** (programmable logic array); **equation** (Boolean logic); or **netlist** (gates). This attribute is read-only and cannot be set by the user.

dont_touch

Identifies designs that are to be excluded from optimization. Values are **true** (the default) or **false**. Designs with the **dont_touch** attribute set to **true** are not modified or replaced during **compile**. Setting **dont_touch** on a design has an effect only when the design is instantiated within another design as a level of hierarchy; setting **dont_touch** on the top-level design has no effect. Set with **set_dont_touch**.

flatten

When set to **true**, determines that a design is to be flattened during **compile**. By default, a design is not flattened. Set with **set_flatten**.

flatten_effort

Defines the level of CPU effort that **compile** uses to flatten a design. Allowed values are **low** (the default), **medium**, or **high**. Set with **set_flatten**.

flatten_minimize

Defines the minimization strategy used for logic equations. Allowed values are **single_output**, **multiple_output**, or **none**. Set with **set_flatten**.

flatten_phase

When **true**, allows logic flattening to invert the phase of outputs during **compile**. By default, logic flattening does not invert the phase of outputs. Used only if the **flatten** attribute is set. Set with **set_flatten**.

is_combinational

true if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute is read-only and cannot be set by the user.

is_dw_subblock

true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute is read-only and cannot be set by the user.

NOTE: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical

true if the design contains leaf cells or other levels of hierarchy. This attribute is read only and cannot be set by the user.

is_mapped

true if all the non-hierarchical cells of a design are mapped to cells in a technology library. This attribute is read-only and cannot be set by the user.

is_sequential

true if any cells of a design or designs in its hierarchy are sequential. A cell is sequential if it is not combinational (if any of its outputs depend on previous inputs). This attribute is read-only and cannot be set by the user.

is_synlib_module

true if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is read-only and cannot be set by the user.

NOTE: synlib modules that are manually elaborated will not have this attribute.

is_unmapped

true if any of the cells are not linked to a design or mapped to a technology library. This attribute is read-only and cannot be set by the user.

local_link_library

A string that contains a list of design files and libraries to be added to the beginning of the **link_library** whenever a **link** operation is performed. Set with **set_local_link_library**.

map_only

When set to **true**, **compile** will attempt to map the object exactly in the target library, and will exclude the object from logic-level optimization (flattening and structuring). The default is **false**. Set with **set_map_only**.

max_capacitance

A floating point number that sets the maximum capacitance value for input, output, or bidirectional ports; or for designs. The units must be consistent with those of the technology library used during optimization. Set with **set_max_capacitance**.

max_dynamic_power

A floating point number that specifies the maximum target dynamic power for the **current_design**. The units must be consistent with those of the technology library. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_dynamic_power**.

max_leakage_power

A floating point number that specifies the maximum target leakage power for the **current_design**. The units must be consistent with those of the technology library. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_leakage_power**.

max_total_power

A floating point number that specifies the maximum target total power for the **current_design**. Total power is defined as the sum of dynamic and leakage power. If this attribute is specified more than once for a design, the latest value is used. Set with **set_max_total_power**.

minimize_tree_delay

When **true** (the default value), **compile** restructures expression trees in the **current_design** or in a list of specified designs, to minimize tree delay. Set this attribute to **false** for any designs that you do not wish to be restructured. Set with **set_minimize_tree_delay**.

model_map_effort

Specifies the relative amount of CPU time to be used by **compile** during modeling, typically for synthetic library implementations. Values are **low**, **medium**, and **high**, or 1, 2, and 3. If **model_map_effort** is not set, the value of **synlib_model_map_effort** is used. Set with **set_model_map_effort**.

model_scale

A floating point number that sets the model scale factor for the **current_design**. Set with **set_model_scale**.

optimize_registers

When **true** (the default value), **compile** automatically invokes the Behavioral Compiler **optimize_registers** command to retime the design during optimization. Setting the attribute to **false** disables this behavior.

Your design cannot contain generic logic at the instant **optimize_registers** is invoked during **compile**. Set with **set_optimize_registers**.

part

A string value that specifies the Xilinx part type for a design. For valid part types, refer to the Xilinx *XC4000 Databook*. Set with **set_attribute**.

resource_allocation

Indicates the type of resource allocation to be used by **compile** for the **current_design**. Allowed values are **none**, indicating no resource sharing; **area_only**, indicating resource sharing with tree balancing without considering timing constraints; **area_no_tree_balancing**, indicating resource sharing without tree balancing and without considering timing constraints; and **constraint_driven** (the default), indicating resource sharing so that timing constraints are met or not worsened. The value of this attribute overrides the value of the variable **hlo_resource_allocation** for the **current_design**. Set with **set_resource_allocation**.

resource_implementation

Indicates the type of resource implementation to be used by **compile** for the **current_design**. Allowed values are **area_only**, indicating resource implementation without considering timing constraints; **constraint_driven**, indicating resource implementation so that timing constraints are met or not worsened; and **use_fastest**, indicating resource implementation using the fastest implementation initially and using components with smaller area later in uncritical parts of the design. The value of this attribute overrides the value of the variable **hlo_resource_implementation** for the **current_design**. Set with **set_resource_implementation**.

structure

Determines if a design is to be structured during **compile**. If **true**, adds logic structure to a design by adding intermediate variables that are factored out of the design's equations. Set with **set_structure**.

structure_boolean

Enables the use of Boolean (non-algebraic) techniques during the structuring phase of optimization. This attribute is ignored if the **structure** attribute is **false**. Set with **set_structure**.

structure_timing

Enables timing constraints to be considered during the structuring phase of optimization. This attribute is ignored if the **structure** attribute is **false**. Set with **set_structure**.

ungroup

Removes a level of hierarchy from the current design by exploding the contents of the specified cell in the current design. Set with **set_ungroup**.

wired_logic_disable

When **true**, disables the creation of wired OR logic during **compile**. The default is **false**; if this attribute is not set, wired OR logic will be created if appropriate. Set with **set_wired_logic_disable**.

wire_load_model_mode

Determines which wire load model to use 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 are **top**, use the wire load model at the top hierarchical level; **enclosed**, use the wire load model on the smallest design that encloses a net completely; and **segmented**, break 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 a value is not specified for this attribute, **compile** searches for a default in the first library in the link path. If none is found, **top** is the default. Set with **set_wire_load**.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)
```

die_area_attributes

Contains attributes related to die area.

DESCRIPTION

Contains attributes related to die area.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class die_area -application**, the definition of attributes can be listed.

Die Area Attributes

bbox

Specifies the bounding-box of a die area. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is read-only.

boundary

Specifies point list of a die area's boundary.

The data type of **boundary** is string.

This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a die area object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

name

Specifies name of a die area object.

The data type of **name** is string.

This attribute is read-only.

object_class

Specifies object class name of a die area, which is **die_area**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

The data type of **object_id** is integer.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

disable_auto_time_borrow

Determines whether the **report_timing** command and other commands will use automatic time borrowing.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

This variable determines whether the **report_timing** command and other commands will use automatic time borrowing. When the value is **false** (the default), automatic time borrowing occurs. Automatic time borrowing balances the slack along back-to-back latch paths, to reduce the overall delay cost. Allocating slack throughout the latch stages can improve optimization results.

When set to **true**, no slack balancing occurs during time borrowing. This means the first paths borrow enough time to meet the constraint until the *max_time_borrow* is reached. Setting the variable to **true** produces time-borrow results consistent with PrimeTime.

SEE ALSO

```
report_timing(2)
timing_variables(3)
```

disable_case_analysis

Disables constant propagation from both logic constants and **set_case_analysis** command constants when set to **true**.

TYPE

Boolean

DEFAULT

false

GROUP

timing

DESCRIPTION

When this variable is set to **true**, it disables constant propagation from both logic constants and **set_case_analysis** command constants. By default, the **disable_case_analysis** variable is **false**, so constant propagation is performed if there has been a **set_case_analysis** command, or if the **case_analysis_with_logic_constants** variable has been specified.

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

SEE ALSO

```
remove_case_analysis(2)
report_case_analysis(2)
set_case_analysis(2)
case_analysis_with_logic_constants(3)
```

disable_library_transition_degradation

Controls whether the transition degradation table is used to determine the net transition time.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

When this variable is set to **false** (the default), **report_timing** and other commands that use timing in dc_shell use the transition degradation table in the library to determine the net transition time. When **true**, the timing commands behave as though there were no transition degradation across the net.

SEE ALSO

```
report_timing(2)
timing_variables(3)
```

do_operand_isolation

Enables or disables operand isolation as a dynamic power optimization technique for a design.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable enables or disables operand isolation as a dynamic power optimization technique for a design. When set to **false** (the default value), this technique is disabled. Set the value to **true** to enable operand isolation.

Operand isolation can insert isolation cells for both operators and hierarchical combinational cells. There are two mechanisms for operand isolation: automatic selection and user-driven mode. This and other parameters can be specified with the **set_operand_isolation_style** command.

To determine the current value of this variable, use the **printvar do_operand_isolation** command. For a list of power variables and their current values, use the **print_variable_group power** command.

SEE ALSO

dont_bind_unused_pins_to_logic_constant

Specify whether the tool should use logic constants to drive all unused pins.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

By default, when loading designs, if there are unused pins (pins without any drivers), the tool automatically creates logic-zero cells and then bind those unused pins to the logic-zero cells.

If the **dont_bind_unused_pins_to_logic_constants** variable is set to **false**, all unused pins are driven by logic constants.

If the **dont_bind_unused_pins_to_logic_constants** variable is set to **true**, the tool does not bind unused pins to logic constants.

The default is **false**.

This variable affects timing case analysis settings.

SEE ALSO

dont_touch_nets_with_size_only_cells

Specifies whether a net is marked as dont_touch when it connects at least one leaf-level size-only driver and at least one leaf-level size-only load.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

When set to **true**, this variable indicates whether nets with one of the following types of connections will have an implicit dont_touch attribute marking. These connections are considered transparently across design hierarchies.

- The net connects a leaf-level size-only driving cell and at least one leaf-level size-only load cell
- The net connects a top-level input port and a leaf-level size-only load cell
- The net connects a leaf-level size-only driving cell and a top-level output port

If a net is explicitly set as **dont_touch false** by using the **set_dont_touch** command, this dont_touch setting supersedes the dont_touch marking derived from the variable.

Cells can be marked as size-only by using the **set_size_only** command.

Be cautious when setting this variable to **true**. If a net is marked as dont_touch, it cannot be changed by commands that optimize or manipulate the design. In particular, dont_touch feedthrough nets and the dont_touch nets connected to two output ports will not be fixed.

The default value of this variable is **false**.

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

SEE ALSO

```
set_size_only(2)
set_dont_touch(2)
report_dont_touch(2)
```

droute_advancedRouteLoops

DESCRIPTION

The **droute_advancedRouteLoops** variable specifies the number of loops for the IC Compiler command **route_advanced**.If the value is set to N, it means that the **route_advanced** command stops after N loops.

This variable applies only to the classic router.

SYNTAX

```
set droute_advancedRouteLoops value
```

```
The valid values of this variable range between 0 and 50.
The default is 3.
```

EXAMPLE

To stop **route_advanced** after 15 loops, enter

```
prompt> set droute_advancedRouteLoops 15
```

droute_areaSrLoop

DESCRIPTION

The **droute_areaSrLoop** variable specifies the number of search and repair loops during area optimization and routing in the **route_area** command. The search and repair loop terminates if the DRC count reaches 0, even if the specified number of iterations are not reached. If the DRC number is still converging (reducing) after reaching the specified number of iterations, specifying more iterations could help to further reduce the DRC count, at the cost of more runtime.

This variable applies only to the classic router.

SYNTAX

```
      set droute_areaSrLoop value

The valid values of this variable range between 0 and 200.
The default is 20.
```

EXAMPLE

To specify the number of search and repair loops during area optimization and routing as 50, enter

set droute_areaSrLoop 50

SEE ALSO

`route_area(2)`

droute_autoSaveInterval

DESCRIPTION

The **droute_autoSaveInterval** variable specifies how frequent checkpoints (intermediate results) are saved during detail routing. If the value is set to 0, no checkpoint is saved during detail routing. Otherwise, the variable sets the time interval in minutes for each checkpoint to be saved.

This variable applies only to the classic router.

SYNTAX

```
      set droute_autoSaveInterval value

The valid values of this variable range between 0 and 14400.
The default is 120.
```

EXAMPLE

To specify how frequent checkpoints (intermediate results) are saved during detail routing, enter

set droute_autoSaveInterval 150

droute_autoSrLoop

DESCRIPTION

The **droute_autoSrLoop** variable sets the number of search and repair loops to be carried out after initial routing.The search and repair loop terminates if a DRC violation count reaches 0 even if the specified number of iterations is not reached. If the DRC violation number is still converging (reducing) after reaching the specified number of iterations, specifying more iterations could further help in reducing the DRC violation count at the cost of more runtime.

This variable applies only to the classic router.

SYNTAX

```
      set droute_autoSrLoop value

The valid values of this variable range between 0 and 500.
The default is 0.
```

EXAMPLE

To set the number of search and repair loops to be carried out after initial routing as 100, enter

set droute_autoSrLoop 100

droute_checkFixedDRC

Controls whether the classic router checks for DRC violations between fixed shapes.

TYPE

Integer. 0 ("no") or 1 ("yes").

DEFAULT

0

DESCRIPTION

By default, the classic router does not check for DRC violations between fixed shapes.

When you set this variable to 1, the classic router checks and reports DRC violations for fixed shapes, in addition to the regular checking of unfixed shapes.

Note that the router always checks for violations between unfixed shapes and fixed shapes.

This variable applies only to the classic router.

EXAMPLE

To enable DRC checking for fixed shapes, enter the following command:

```
prompt> set_app_var droute_checkFixedDRC 1
```

SEE ALSO

`verify_route(2)`

droute_connBrokenNet

DESCRIPTION

The **droute_connBrokenNet** variable specifies whether broken nets are to be left broken or are to be reconnected. A broken net is a net that is not totally connected by glinks, preroutes, wires, or vias. If the value is set to 0, broken nets are ignored during detail routing. If the value is set to 1, broken nets are to be reconnected by detail routing.This variable does not affect tie-off connections, which are controlled by **the connTieOff** variable.

This variable applies only to the classic router.

SYNTAX

```
        set droute_connBrokenNet value

   When set to 1 (the default), the classic router reconnects broken
nets.
   When set to 0, the classic router ignores broken nets.
```

EXAMPLE

To ignore broken nets, enter

set droute_connBrokenNet 0

When set to 1, the broken nets are reconnected.

droute_connTieOff

Controls whether the classic router connects tie-off nets.

TYPE

Integer. 0 ("no") or 1 ("yes").

DEFAULT

1

DESCRIPTION

By default, the classic router connects tie-off nets.

When you set this variable to 0, the classic router does not connect tie-off nets.

This variable applies only to the classic router.

EXAMPLE

To disable the connection of tie-off nets, enter the following command:

```
prompt> set_app_var droute_connTieOff 0
```

SEE ALSO

```
route_area(2)
route_auto(2)
route_detail(2)
route_eco(2)
route_group(2)
route_opt(2)
route_search_repair(2)
droute_maxTieOffDistance(3)
```

droute_connTieRail

DESCRIPTION

The **droute_connTieRail** variable specifies where to connect tie-off pins. If the value is set to 0, the router connects tie-off pins anywhere on power and ground nets. If the value is set to 1, the router connects tie-off pins to power and ground rails and power and ground pins only.

This variable applies only to the classic router.

SYNTAX

```
        set droute_connTieRail value

   When set to 0 (the default), the classic router connects tie-off pins
   anywhere.
   When set to 1, the classic router connects tie-off pins only to a
power
   or ground rail and to power or ground pins.
```

EXAMPLE

To connect tie pins only to power and ground rails and power and ground pins, enter

set droute_connTieRail 1

When set to 0, the tie-off pins can be connected anywhere.

droute_doMaxCapConx

DESCRIPTION

The **droute_doMaxCapConx** variable specifies whether maximum capacitance constraints should be met or ignored during in-route optimization. If the value is set to 0, the maximum capacitance constraints are ignored during in-route optimization. If the value is set to 1, in-route optimization tries to meet maximum capacitance constraints.

This variable applies only to the classic router.

SYNTAX

```
set droute_doMaxCapConx value
```

EXAMPLE

To ignore maximum capacitance constraints, enter

set droute_doMaxCapConx 0

droute_doMaxTransConx

DESCRIPTION

The **droute_doMaxTransConx** variable specifies whether maximum transition constraints should be met or ignored during in-route optimization. If the value is set to 0, the maximum transition constraints are ignored during in-route optimization. If the value is set to 1, in-route optimization tries to meet maximum transition constraints.

This variable applies only to the classic router.

SYNTAX

```
set droute_doMaxTransConx value
```

EXAMPLE

To ignore maximum transition constraints, enter

set droute_doMaxTransConx 0

droute_doProbeConx

DESCRIPTION

The **droute_doProbeConx** variable specifies whether nets with top-layer probe constraints are to be routed to top layer for probing.

This variable applies only to the classic router.

SYNTAX

```
        set droute_doProbeConx value

   When set to 0 (the default), the classic router ignores top-layer
probe
   constraints.
   When set to 1, the classic router routes nets with top-layer probe
constraints
   on the top layer for probing.
```

EXAMPLE

To route the nets with top-layer probe constraints to the top layer for probing, enter

set droute_doProbeConx 1

droute_doSelectedNetAntennaConx

DESCRIPTION

The **droute_doSelectedNetAntennaConx** variable controls whether all antenna violations should be fixed or whether only antenna violations for specified nets should be fixed.

This variable applies only to the classic router.

SYNTAX

```
        set droute_doSelectedNetAntennaConx value

   When set to 0 (the default), the classic router fixes all antenna
violations.
   When set to 1, the classic router fixes antenna violations only for
the
   specified nets.
```

EXAMPLE

To fix antenna violation for specified nets only, enter

set droute_doSelectedNetAntennaConx 1

droute_doXtalkConx

DESCRIPTION

The **droute_doXtalkConx** variable specifies whether crosstalk constraints (such as static noise voltage constraints and capacitance constraints) are to be honored during in-route optimization.

This variable applies only to the classic router.

SYNTAX

```
set droute_doXtalkConx value
```

EXAMPLE

To honor cross-talk constraints, enter

set droute_doXtalkConx 1

droute_ecoListToFile

DESCRIPTION

The **droute_ecoListToFile** variable specifies whether to save the routing ECO list to a file named ecoRoute.list.

This variable applies only to the classic router.

SYNTAX

```
        set droute_ecoListToFile value

   When set to 0 (the default), the classic router does not save the
routing
   ECO list to a file, however, you can still find it in the log file.
   When set to 1, the routing ECO list is output to the ecoRoute.list
file.
```

EXAMPLE

To dump a list of modified nets to a file named ecoRoute.list, enter

set droute_ecoListToFile 1

droute_ecoMode

DESCRIPTION

The **droute_ecoMode** variable specifies what nets are to be routed in ECO mode, and how they are rerouted. If the value is set to 0, only modified nets can be rerouted. If the value is set to 1, modified nets are rerouted first, and other nets can be rerouted later. If the value is set to 2 (the default), any nets can be rerouted in any order. Mode 0 produces minimum routing change, but offers you the least freedom to resolve DRC violations. Mode 2 offers the most freedom to resolve DRC violations, but can introduce significant changes in the routing solution.

This variable applies only to the classic router.

SYNTAX

```
set droute_ecoMode value
```

EXAMPLE

To reroute modified nets only, enter

set droute_ecoMode 0

droute_ecoScope

DESCRIPTION

The **droute_ecoScope** variable specifies whether to use global or local scope during ECO routing. If the value is set to 0 (the default), the router uses global scope mode. If the value is set to 1, the router uses local scope. Set this value to 0 if design has many violations. Set it to 1 to get better runtime if the design has very few violations.

This variable applies only to the classic router.

SYNTAX

```
set droute_ecoScope value
```

EXAMPLE

To use local scope if there are very few localized violations, enter

set droute_ecoScope 1

droute_ecoSrLoop

DESCRIPTION

The **droute_ecoSrLoop** variable sets the number of search and repair loops to be carried out after ECO routing. The search and repair loop terminates if the DRC violation count reaches 0, even if the specified number of iterations are not reached. If the DRC violation number is still converging (reducing) after reaching the specified number of iterations, specifying more iterations could help further reduce DRC violation count at the cost of more runtime.

This variable applies only to the classic router.

SYNTAX

```
      set droute_ecoSrLoop value

The valid values of this variable range between 0 and 500.
The default is 20.
```

EXAMPLE

To set the number of search and repair loops to be carried out after ECO routing as 90, enter the following:

set droute_ecoSrLoop 90

droute_enable_one_pass_partitioning

Specifies which partitioning algorithm to use for distributed routing.

TYPE

integer

DEFAULT

0

DESCRIPTION

The **droute_enable_one_pass_partitioning** variable specifies which partitioning algorithm to use for distributed processing of routing. The original partitioning algorithm is used when the variable is set to 0. The newly designed partitioning algorithm is used when the variable is set to 1. The one-pass partitioning algorithm improves the runtime of distributed processing of detail routing on large designs as compared to the default two-pass partitioning algorithm.

The valid value of the variable is 0 or 1. The default value is 0.

This variable applies only to the classic router.

EXAMPLE

To enable the new partitioning algorithm, enter the following command:

```
prompt> set droute_enable_one_pass_partitioning 1
```

droute_expandFillTracks

DESCRIPTION

The **droute_expandFillTracks** variable specifies whether the track region for metal fill should be expanded or not.

When set to 0 (the default), the router inserts metal fills only in the core area. When set to 1, the router expands the track region for metal fill to cover the whole design area.

This variable applies only to the classic router.

SYNTAX

```
set droute_expandFillTracks value
```

EXAMPLE

To enable the router to expand the track region to cover the entire design area for metal fill, enter

set droute_expandFillTracks 1

droute_fillDataType

DESCRIPTION

The **droute_fillDataType** variable specifies a data type of filled metals. The value N is the data type of filled metals.

This variable applies only to the classic router.

SYNTAX

```
      set droute_fillDataType value

The valid values of this variable range between 0 and 256.
The default is 0.
```

EXAMPLE

To specify the data type of filled metals as 90, enter

set droute_fillDataType 90

droute_fillEndByMinSpcPercent

DESCRIPTION

The **droute_fillEndByMinSpcPercent** variable specifies the spacing requirement between the routing wires and the ends of fill metal. If the value is set to -1, the normal required spacing is used between the routing wires and the ends of fill metal. Otherwise, the value sets the percentage of normal required spacing to be used between the routing wires and the ends of fill metal. Normally, this value is to be left at its default value of -1. It should only be changed if the user understands the impact of it on the yield, and the final verification should be set accordingly.

This variable applies only to the classic router.

SYNTAX

```
      set droute_fillEndByMinSpcPercent value

The valid values of this variable range between -1 and 10,000.
The default is -1.
```

EXAMPLE

To use the same spacing for all sides between fill metal and routing wires as 500, enter

set droute_fillEndByMinSpcPercent 500

droute_fillMetalCloseToMinDensityValue

DESCRIPTION

The **droute_fillMetalCloseToMinDensityValue** variable specifies the target metal density for metal fill insertion. If the value is set to 0 (the default), the router tries to maximize the dummy metal fill as long as the maximum density is not violated. If the value is set to 1, the router tries to minimize dummy metal fill as long as the minimum density is met.

This variable applies only to the classic router.

SYNTAX

```
set droute_fillMetalCloseToMinDensityValue value
```

EXAMPLE

For fill metal to have metal density a little more than minimum density value, enter

set droute_fillMetalCloseToMinDensityValue 1

droute_fillMetalUniformly

DESCRIPTION

The **droute_fillMetalUniformly** variable specifies whether to fill dummy metal uniformly. If the value is set to 0 (the default), the router fills dummy metal in contiguous tracks. If the value is set to 1, the router tries to fill dummy metal uniformly in each window to meet the density rule.

This variable applies only to the classic router.

SYNTAX

```
set droute_fillMetalUniformly value
```

EXAMPLE

To specify fill metal uniformly in each window to meet the density rule, enter

set droute_fillMetalUniformly 1

droute_fillViaDataType

DESCRIPTION

The **droute_fillViaDataType** variable specifies a data type of filled vias. If the value set to N, N is the data type of filled via.

This variable applies only to the classic router.

SYNTAX

```
      set droute_fillViaDataType value

The valid values of this variable range between 0 and 256.
The default is 0.
```

EXAMPLE

To specify the data type of filled vias, enter

set droute_fillViaDataType 70

droute_fixIsoViaTouchClock

Controls whether the classic router can use vias that touch clock nets when fixing isolated via DRC violations.

TYPE

Integer. 0 ("no") or 1 ("yes").

DEFAULT

1

DESCRIPTION

The classic router fixes isolated via DRC violations by adding vias. The added vias might touch other nets that are not associated with the original isolated via. This variable controls whether the router can add vias that touch clock nets.

By default, the classic router can use vias that touch clock nets.

When you set this variable to 0, the classic router cannot add vias that touch clock nets when fixing isolated via DRC violations.

This variable applies only to the classic router.

EXAMPLE

To prevent the classic router from adding vias that touch clock nets when fixing isolated via DRC violations, enter the following command:

```
prompt> set_app_var droute_fixIsoViaTouchClock 0
```

SEE ALSO

```
route_area(2)
route_auto(2)
route_detail(2)
route_eco(2)
route_group(2)
```

```
route_opt(2)
route_search_repair(2)
droute_fixIsoViaTouchPG(3)
```

droute_fixIsoViaTouchPG

Controls whether the classic router can use vias that touch power or ground nets when fixing isolated via DRC violations.

TYPE

Integer. 0 ("no") or 1 ("yes").

DEFAULT

1

DESCRIPTION

The classic router fixes isolated via DRC violations by adding vias. The added vias might touch other nets that are not associated with the original isolated via. This variable controls whether the router can add vias that touch power or ground nets.

By default, the classic router can use vias that touch power or ground nets.

When you set this variable to 0, the classic router cannot add vias that touch power or ground nets when fixing isolated via DRC violations.

This variable applies only to the classic router.

EXAMPLE

To prevent the classic router from adding vias that touch power or ground nets when fixing isolated via DRC violations, enter the following command:

```
prompt> set_app_var droute_fixIsoViaTouchPG 0
```

SEE ALSO

```
route_area(2)
route_auto(2)
route_detail(2)
route_eco(2)
```

```
route_group(2)
route_search_repair(2)
route_opt(2)
droute_fixIsoViaTouchClock(3)
```

droute_fixMinEdgeLengthByFilling

DESCRIPTION

The **droute_fixMinEdgeLengthByFilling** variable specifies whether to allow filling metal to fix non-pin minimum edge length violations during search and repair.

When set to 0 (the default), the router cannot use filling to resolve minimum edge length violations during search and repair. When set to 1, the router can use filling to resolve minimum edge length violations during search and repair.

This variable applies only to the classic router.

SYNTAX

```
set droute_fixMinEdgeLengthByFilling value
```

EXAMPLE

To allow filling to fix non-pin minimum edge length violations during search and repair, enter

set droute_fixMinEdgeLengthByFilling 1

droute_followPolyTrkForPolyFill

DESCRIPTION

The **droute_followPolyTrkForPolyFill** variable specifies to whether tracks on the poly layer are used for poly fills. If the value is set to 0 (the default), the router follows M2 tracks to fill the poly layer. If the value is set to 1, the router follows poly tracks to fill the poly layer.

This variable applies only to the classic router.

SYNTAX

```
set droute_followPolyTrkForPolyFill value
```

EXAMPLE

To specify whether the router should follow poly tracks for metal fill on the poly layer, enter

set droute_followPolyTrkForPolyFill 1

droute_groupSrLoop

DESCRIPTION

The **droute_groupSrLoop** variable specifies the number of search and repair loops in the **route_group** command. The search and repair loop terminates if the DRC violation count reaches 0, even if the specified number of iterations is not reached. If the DRC violation number is still converging (reducing) after reaching specifies number of iterations, specifying more iterations could help to further reduce the DRC violation at the cost of more runtime.

This variable applies only to the classic router.

SYNTAX

```
     set droute_groupSrLoop value
```

```
The valid values of this variable range between 0 and 200.
The default is 5.
```

EXAMPLE

To specify five search and repair loops in the **route_group** command, enter

set droute_groupSrLoop 5

droute_highEffortViaDoubling

Controls the method used by the classic router to connect double vias to pins.

TYPE

Integer. 0 or 1.

DEFAULT

1

DESCRIPTION

Insertion of double vias is done by the classic router **insert_redundant_vias** command.

A double via can connect to a pin either by being fully covered by the pin shape or by only touching the pin, where part of the double via is outside the pin.

Fully covering the pin shape has the advantage of less potential DRC violations. On the other hand, a higher rate of via doubling can be achieved by allowing partial covering of the pin shape.

By default, the **droute_highEffortViaDoubling** variable is set to 1, which allows partial covering of the double via connections to pins.

This variable applies only to the classic router.

EXAMPLE

To require fully covered double via connections to pins, use the following command:

```
prompt> set_app_var droute_highEffortViaDoubling 0
```

SEE ALSO

`insert_redundant_vias(2)`

droute_lowSkewClkRoute

DESCRIPTION

The **droute_lowSkewClkRoute** variable specifies how many changes are allowed on nets from clock tree synthesis during detail routing. If the value is set to 0, nets from clock tree synthesis are treated the same as other nets. If the value is set to 1 (the default), minimal changes are to be made to nets from clock tree synthesis, if necessary.

This variable applies only to the classic router.

SYNTAX

```
set droute_lowSkewClkRoute value
```

EXAMPLE

For normal routing, enter

set droute_lowSkewClkRoute 0

droute_maxOffGridTrack

DESCRIPTION

The **droute_maxOffGridTrack** variable specifies how off-grid tracks are to be added for off-grid pins and off-grid feedthroughs. If the value is set to 0 (the default), off-grid tracks are added selectively in via regions, which are determined by the router. If the value is set to 1, as many off-grid tracks as possible are added for off-grid pins. If the value is set to 2, 3, or 4, more off-grid tracks are aggressively added in increasing order for off-grid feedthroughs and pins. By creating as many off-grid tracks as possible for off-grid pins, the router has the maximum flexibility in accessing these pins, at the cost of more routing time. You should leave the **droute_maxOffGridTrack** variable at the default value of 0 to start with if the you do not know how bad the off-grid situation is, and gradually increase the value if the results suffer from off-grid pin access or off-grid feedthrough usage. If you know from previous experience that there will be off-grid difficulty, you can set the **droute_maxOffGridTrack** variable to a higher value to start with. You can also set the value to -1. This lets the router start from 0 and then automatically and gradually increase to 1, 2, 3, 4 during search and repair. The side effect of setting the **droute_maxOffGridTrack** variable to a higher value is increased runtime.

This variable applies only to the classic router.

SYNTAX

```
set droute_maxOffGridTrack value
```

EXAMPLE

To selectively add off-grid tracks, enter

set droute_maxOffGridTrack 0

droute_maxTieOffDistance

DESCRIPTION

The **droute_maxTieOffDistance** variable specifies the search distance to connect tieoffs. If the value is set to -1, there is no search distance limit to connect tieoffs. Otherwise, the variable specifies the search distance in number of global routing cells for the search engine to use to connect tieoffs.

This variable applies only to the classic router.

SYNTAX

```
      set droute_maxTieOffDistance value

   The valid values of this variable range between -1 and 20,000.
   The default is 10.
```

EXAMPLE

To specify no limit on the search distance, enter

set droute_maxTieOffDistance -1

droute_metalFillDensityIncrement

DESCRIPTION

The **droute_metalFillDensityIncrement** variable specifies the increment value for metal fill. If the value is set to N, the router targets metal density N percent higher than the minimum density during dummy metal insertion. The value N specifies the increment from the **minDensity** value for metal fill. It is used when **fillMetalCloseToMinDensityValue** is enabled. For example, if the **minDensity** value specified in the technology file is 20 percent and **droute_metalFillDensityIncrement** is set to 10 percent, the target density for metal fill is 30 percent.

This variable applies only to the classic router.

SYNTAX

```
      set droute_metalFillDensityIncrement value

The valid values of this variable range between 0 and 100.
The default is 10.
```

EXAMPLE

To specify increment value for metal fill as 50, enter

set droute_metalFillDensityIncrement 50

droute_minLengthCheckCutMode

DESCRIPTION

The **droute_minLengthCheckCutMode** variable specifies how to compute minimum length when a polygon completely encloses a via. If the value is set to 0 (the default), the router does not check whether a wire completely encloses via. If the value is set to 1, the router checks whether a wire completely encloses via. When a via connects more than one wire segment on the same metal layer, minimum length is checked against the longest wire that completely encloses the via. When a via connects only to single wire segment on one metal layer, the wire needs to be checked for minimum length rule.

This variable applies only to the classic router.

SYNTAX

```
set droute_minLengthCheckCutMode value
```

EXAMPLE

To check overlapping cuts on less than minimum length segments, enter

set droute_minLengthCheckCutMode 1

When set to 0, the overlapping cuts are not checked.

droute_minShieldLength

DESCRIPTION

The **droute_minShieldLength** variable specifies minimum length for a wire to be shielded in the unit of pitches. For example, if the variable is left at its default value of 4, only the wires longer than 4 pitches are shielded.

This variable applies only to the classic router.

SYNTAX

```
      set droute_minShieldLength value

The valid values of this variable range between 0 and 100.
The default is 4.
```

EXAMPLE

To specify the minimum length for a wire to be shielded as 20, enter

set droute_minShieldLength 20

droute_numCPUs

DESCRIPTION

The **droute_numCPUs** variable specifies the number of CPUs to be used for distributed routing.

This variable applies only to the classic router.

SYNTAX

```
      set droute_numCPUs value
```

```
The valid values of this variable range between 0 and 63.
The default is 0.
```

EXAMPLE

To use a single CPU mode, enter

set droute_numCPUs 1

droute_offGridCost

DESCRIPTION

The **droute_offGridCost** variable specifies extra off-grid routing cost. The default on-grid unit wire cost is 1. Therefore, if the value is set to 0, off-grid routes have the same cost as on-grid cost, and wires are routed as in a gridless router. If the value is set to N, off grid unit wire cost becomes 1 + N.This variable can be used to encourage or discourage off-grid routing against on-grid routing.

This variable applies only to the classic router.

SYNTAX

```
      set droute_offGridCost value
```

```
The valid values of this variable range between 0 and 5.
The default is 1.
```

EXAMPLE

To route as a gridless router, enter

set droute_offGridCost 0

droute_offsetFillTrack

DESCRIPTION

The **droute_offsetFillTrack** variable specifies how to offset the metal fill track.

When set to 0 (the default), the router automatically determines the initial track for metal fill. When set to 1, the router offsets the initial track by half of the fill-pitch.

This variable applies only to the classic router.

SYNTAX

```
set droute_offsetFillTrack value
```

EXAMPLE

To enable the router to offset the initial track by half fill-pitch, enter

set droute_offsetFillTrack 1

droute_optDelaySlackTarget

DESCRIPTION

The **droute_optDelaySlackTarget** variable specifies slack target for delay optimization during in-route optimization.

This variable applies only to the classic router.

SYNTAX

```
      set droute_optDelaySlackTarget value
```

```
The valid values of this variable range between -100.000
and 100.000. The default is 0.100.
```

EXAMPLE

To specify slack target for delay optimization during in-route optimization as 90, enter

set droute_optDelaySlackTarget 90.000

droute_optDelaySrLoop

DESCRIPTION

The **droute_optDelaySrLoop** variable specifies the number of search and repair loops after delay optimization in the **route_opt** command. The search and repair loop terminates if the DRC violation count reaches 0, even if the specified number of iterations is not reached. If the DRC violation number is still converging (reducing) after reaching the specified number of iterations, specifying more iterations could help to further reduce the DRC violation count at the cost of more runtime.

This variable applies only to the classic router.

SYNTAX

```
      set droute_optDelaySrLoop value
```

```
The valid values of this variable range between 0 and 200.
The default is 5.
```

EXAMPLE

To specify the number of search and repair loops as 5 after delay optimization in the **route_opt** command, enter the following:

set droute_optDelaySrLoop 5

droute_optSetup

DESCRIPTION

The **droute_optSetup** variable specifies whether setup slack is to be maintained or improved during in-route optimization. If the value is set to 0, the router tries to maintain setup slack, and does not try to improve the slack. If the value is set to 1 (the\ default), the router tries to improve setup slack.

This variable applies only to the classic router.

SYNTAX

```
set droute_optSetup value
```

EXAMPLE

To maintain setup slack, enter

set droute_optSetup 0

droute_optSrLoop

DESCRIPTION

The **droute_optSrLoop** variable specifies the number of search and repair loops after route optimization in the **route_opt** command. The search and repair loop terminates if the DRC violation count reaches 0, even if the specified number of iterations is not reached. If the DRC violation number is still converging (reducing) after reaching specified number of iterations, specifying more iterations could help to further reduce the DRC violation count at the cost of more runtime.

This variable applies only to the classic router.

SYNTAX

```
      set droute_optSrLoop value

The valid values of this variable range between 0 and 200.
The default is 20.
```

EXAMPLE

To specify the number of search and repair loops after route optimization as 20, enter

set droute_optSrLoop 20

droute_optViaHoldTimeThreshold

DESCRIPTION

The **droute_optViaHoldTimeThreshold** variable specifies the hold time threshold for contact optimization. If the value is set to N, the router tries to preserve hold time by not inserting redundant vias for the nets whose hold time is worse than N.

This variable applies only to the classic router.

SYNTAX

```
      set droute_optViaHoldTimeThreshold value

The valid values of this variable range between -1000000.000
and 1000000.000. The default is 0.000.
```

EXAMPLE

To specify the hold time threshold for contact optimization as 800, enter

set droute_optViaHoldTimeThreshold 800.000

set droute_optViaHoldTimeThreshold

droute_optViaSetupSlackThreshold

DESCRIPTION

The **droute_optViaSetupSlackThreshold** variable specifies the setup slack threshold for contact optimization. If the value is set to N, the router tries to preserve setup time by not inserting redundant vias for the nets whose setup slack is worse than N.

This variable applies only to the classic router.

SYNTAX

```
     set droute_optViaSetupSlackThreshold value

The valid values of this variable range between -1000000.000
and 1000000.000. The default is -0.100.
```

EXAMPLE

To specify the setup slack threshold for contact optimization as -800, enter

set droute_optViaSetupSlackThreshold

set droute_optViaSetupSlackThreshold

droute_optViaSrLoop

DESCRIPTION

The **droute_optViaSrLoop** variable specifies search and repair loops after contact optimization. If the value set to N, the router runs N loops of search and repair after contact optimization.

This variable applies only to the classic router.

SYNTAX

```
      set droute_optViaSrLoop value

The valid values of this variable range between 0 and 100.
The default is 1.
```

EXAMPLE

To specify search and repair loops after contact optimization as 40, enter

set droute_optViaSrLoop 40

droute_optViaTimingDriven

DESCRIPTION

The **droute_optViaTimingDriven** variable specifies whether to preserve timing for critical nets with timing violations. If the value is set to 0 (the default), the router does contact optimization for all nets. If the value is set to 1, the router tries to preserve timing for critical nets by not doing contact optimization on those nets.

This variable applies only to the classic router.

SYNTAX

```
set droute_optViaTimingDriven value
```

EXAMPLE

To perform contact optimization for all nets, enter the following:

set droute_optViaTimingDriven 0

droute_optimizeRouteGroup

DESCRIPTION

The **droute_optimizeRouteGroup** variable specifies whether optimization is to be run in the **route_group** command. Skipping optimization at the end of net group routing could save some runtime.

This variable applies only to the classic router.

SYNTAX

```
set droute_optimizeRouteGroup value
```

EXAMPLE

To skip optimization at the end of the **route_group** command, enter

set droute_optimizeRouteGroup 0

droute_parallelLengthMode

DESCRIPTION

The **droute_parallelLengthMode** variable specifies how to compute parallel length for the fat metal spacing rule. If the value is set to 0 (the default), the router merges only fat neighboring shapes and determines parallel length based on the merged fat wire. If the value is set to 1, the router merges all neighboring shapes (fat and thin) and the total length of the merged wire is used to compute the parallel length.

This variable applies only to the classic router.

SYNTAX

```
set droute_parallelLengthMode value
```

EXAMPLE

To merge all neighboring shapes, enter

set droute_parallelLengthMode 1

When set to 0, the router merges only fat neighboring shapes.

droute_pinTaperLengthLimit

DESCRIPTION

The **droute_pinTaperLengthLimit** variable specifies the limit for pin tapering length. If the value is set to -1, there is no limit for tapering length, and tapering should go all the way to the Steiner point. Otherwise, this variable specifies the tapering length limit in the units of routing pitches.

This variable applies only to the classic router.

SYNTAX

```
set droute_pinTaperLengthLimit value
```

```
The valid values of this variable range between -1 and 1,000,000. The
default is 10.
```

EXAMPLE

To specify the limit for pin tapering length as 20, enter

set droute_pinTaperLengthLimit 20

droute_pinTaperMode

DESCRIPTION

The **droute_pinTaperMode** variable specifies the pin tapering width. If the value is set to 0, the tapering width is the default width. If the value is set to 1 (the default), the tapering width is the pin width.

This variable applies only to the classic router.

SYNTAX

```
set droute_pinTaperMode value
```

EXAMPLE

To specify tapering width as the default width, enter

set droute_pinTaperMode 0

droute_reportLimit

DESCRIPTION

The **droute_reportLimit** variable specifies the maximum number of DRC violations that the router reports.If the value is set to -1, no limit is set, and the router reports all DRC violations. Otherwise, the value specifies the maximum number of DRC violations to be reported.

This variable applies only to the classic router.

SYNTAX

```
      set droute_reportLimit value

The valid values of this variable range between -1 and 10,000.
The default is 200.
```

EXAMPLE

To report 500 DRC violations, enter

set droute_reportLimit 500

droute_rerouteUserWire

DESCRIPTION

The **droute_rerouteUserWire** variable specifies whether user-created wires and vias are treated as fixed or reroutable.

When this variable is set to 0 (the default), the router treats user-created wires and vias, such as wires created in the layout editor, as fixed. When this variable is set to 1, the router treats user-created wires and vias as fixed.

This variable applies only to the classic router.

SYNTAX

```
set droute_rerouteUserWire value
```

EXAMPLE

To reroute user-created wires, enter

set droute_rerouteUserWire 1

droute_rerunDRC

DESCRIPTION

The **droute_rerunDRC** variable specifies whether DRC violations are regenerated before search and repair.

When set to 0 (the default), DRC violations from previous results are used by search and repair. When set to 1, DRC violations are regenerated before search and repair.

This variable applies only to the classic router.

SYNTAX

```
set droute_rerunDRC value
```

EXAMPLE

To recheck violations before search and repair, enter

set droute_rerunDRC 1

droute_resetMinMaxLayer

DESCRIPTION

The **droute_resetMinMaxLayer** variable resets the minimum and maximum rule constraints defined by the **set_ignored_layers**, **set_net_routing_layer_constraints**, and **set_net_routing_rule** commands before search and repair.

This variable applies only to the classic router.

SYNTAX

```
set droute_resetMinMaxLayer value
```

EXAMPLE

To reset the layer of all wires based on the minimum and maximum rules before starting search and repair, enter

set droute_resetMinMaxLayer 1

droute_shieldAvoidFatViaArray

Controls the shape of the fat via array that the classic router uses for shielding.

TYPE

Integer. 0 ("no") or 1 ("yes").

DEFAULT

0

DESCRIPTION

By default, the classic router uses rectangular fat via arrays, depending on the requested number of cuts.

When you set this variable to 1, the classic router uses only narrow via arrays (Nx1 or 1xN) for shielding.

This variable applies only to the classic router.

EXAMPLE

To force the use of narrow via arrays for shielding, enter the following command:

```
prompt> set_app_var droute_shieldAvoidFatViaArray 1
```

SEE ALSO

`create_auto_shield(2)`

droute_shieldLimitSboxExt

DESCRIPTION

The **droute_shieldLimitSboxExt** variable specifies the processing of routing area Sboxes during shielding, and is used to speed shielding up by limiting the routing area Sbox extension size to the technology file layer minSpacing parameters in such routing areas.

When set to 0 (the default), the SBox extension size is maximized by the technology file fat layer parameters, which in some cases produces more accurate results, but contributes to increased shielding runtime. When set to 1, the SBox extension size is limited to the technology file layer minSpacing, which reduces the runtime.

This variable applies only to the classic router.

SYNTAX

```
set droute_shieldLimitSboxExt value
```

EXAMPLE

To speed shielding up by limiting routing area Sbox extension size, enter

set droute_shieldLimitSboxExt 1

droute_shieldRerouteSignalNets

DESCRIPTION

The **droute_shieldRerouteSignalNets** variable specifies whether to reroute signal nets during shielding. If the value is set to 0 (the default), the router does not reroute signal nets to improve shielding coverage. If the value is set to 1, the router reroutes signal nets and possibly creates minimized DRC violations as a tradeoff for improved shielding coverage. You need to run search and repair after shielding if DRC violations are created on signal nets.

This variable applies only to the classic router.

SYNTAX

```
set droute_shieldRerouteSignalNets value
```

EXAMPLE

To enable the router reroute signal nets and possibly create minimized DRC violations as a trade-off of improved shielding coverage, enter

set droute_shieldRerouteSignalNets 1

droute_shieldSkipNotShieldedSboxes

DESCRIPTION

The **droute_shieldSkipNotShieldedSboxes** variable specifies the processing of routing area Sboxes that have no shielding, and is used to speed shielding up by skipping checking for DRC violations in such routing areas.

When set to 0 (the default), the router checks DRCs in all Sboxes, even those with no shielding. When set to 1, the router skips DRCs in Sboxes with no shielding to reduce runtime.

This variable applies only to the classic router.

SYNTAX

```
set droute_shieldSkipNotShieldedSboxes value
```

EXAMPLE

To speed shielding up by skipping routing area Sboxes that have no shielding, enter

set droute_shieldSkipNotShieldedSboxes 1

droute_shieldViaMinSpacing

DESCRIPTION

The **droute_shieldViaMinSpacing** variable specifies the spacing requirement for shielded vias. If the value is set to 0 (the default), the spacing specified for shielding wires is used to shield vias. If the value is set to 1, the minimum spacing is used to shield vias to minimize routing resource usage.

This variable applies only to the classic router.

SYNTAX

```
set droute_shieldViaMinSpacing value
```

EXAMPLE

To shield vias with minimum spacing only to minimize overhead, enter

set droute_shieldViaMinSpacing 1

droute_smallJogMinLength

DESCRIPTION

The **droute_smallJogMinLength** variable specifies the minimum length of a small jog. If the value is set to 0, the router does not consider the small jog recommended rule. If the value is set to N, the router reports and fixes the small jog violation if the jog length is less than (N) times a quarter pitch.

This variable applies only to the classic router.

SYNTAX

```
set droute_smallJogMinLength value

The valid values of this variable range between 0 and 2.
The default is 0.
```

EXAMPLE

To specify the minimum length of a small jog as 1, enter

```
prompt> set droute_smallJogMinLength 1
```

droute_srLoop

DESCRIPTION

The **droute_srLoop** variable sets the number of search and repair loops in the search and repair phase in the **route_search_repair** command. The search and repair loop terminates if the DRC violation count reaches 0, even if the specified number of iterations is not reached. If the DRC violation number is still converging (reducing) after reaching the specified number of iterations, specifying more iterations could help to further reduce the DRC violation count at the cost of more runtime.

This variable applies only to the classic router.

SYNTAX

```
      set droute_srLoop value

The valid values of this variable range between 0 and 1000.
The default is 50.
```

EXAMPLE

To set the number of search and repair loops as 50 in the search and repair phase in **axgSearchRepair** command, enter

set droute_srLoop 50

droute_stopIfNoLicense

DESCRIPTION

The **droute_stopIfNoLicense** variable flags whether to stop the router or continue if the tool cannot obtain the option license. If the value is set to 0 (the default), the router continues by ignoring certain constraints if the router fails to get the corresponding option license. If the value is set to 1, the router exits if it fails to get the needed option license.

This variable applies only to the classic router.

SYNTAX

```
set droute_stopIfNoLicense value
```

EXAMPLE

To ensure that the router exits if it fails to get the needed option license, enter

set droute_stopIfNoLicense 1

droute_suppressIllegalContactMsg

Controls whether the classic router issues error messages when illegal contacts are used.

TYPE

Integer. 0 ("no") or 1 ("yes").

DEFAULT

0

DESCRIPTION

An illegal contact is a contact without a legal contact code (via master).

When the classic router reads an illegal contact, it issues an error message. You can suppress this error message by setting the **droute_suppressIllegalContactMsg** variable to 1.

This variable applies only to the classic router.

EXAMPLE

To suppress illegal contact messages, enter the following command:

```
prompt> set_app_var droute_suppressIllegalContactMsg 1
```

SEE ALSO

```
route_detail(2)
route_search_repair(2)
route_group(2)
route_area(2)
route_eco(2)
route_auto(2)
route_opt(2)
verify_route(2)
```

droute_timeLimit

DESCRIPTION

The **droute_timeLimit** variable specifies the corresponding detail routing step runtime limit. If the value is set to -1, no runtime limit is set. Otherwise, the variable sets the runtime limit in minutes. This is a very useful variable if you have a time limit and do not care about the final number of DRC violations as much. For example, you might want to get the feeling of how "routable" a design is in a given amount of time. Not all detail routing steps support time limits, such as **route_auto** and **route_global**. You can check the GUI form to see which routing steps support time limits.

This variable applies only to the classic router.

SYNTAX

```
      set droute_timeLimit value

The valid values of this variable range between -1 and 14400.
The default is -1.
```

EXAMPLE

To specify the corresponding detail routing step runtime limit as 1000, enter

set droute_timeLimit 1000

droute_timingDriven

DESCRIPTION

The **droute_timingDriven** variable specifies whether timing-critical nets with setup violations are to be optimized during initial detail routing. If the value is set to 0 (the default), timing-critical nets with setup violations are not optimized. If the value is set to 1, timing-critical nets with setup violations are optimized. When you set the value to 1, initial detail routing tries to generate routes with less resistance and capacitance (shorter wires and fewer vias) for timing-critical nets.

This variable applies only to the classic router.

SYNTAX

```
set droute_timingDriven value
```

EXAMPLE

For regular routing, enter

set droute_timingDriven 1

droute_timingSpace

DESCRIPTION

The **droute_timingSpace** variable specifies whether the detailed router should try to allocate extra same-layer spacing for timing-critical net wires.

When set to 0 (the default), the router does not allocate extra same-layer spacing for timing-critical nets. When set to 1, the router allocates extra same-layer spacing for timing-critical nets to reduce the intralayer capacitance.

This variable applies only to the classic router.

SYNTAX

```
set droute_timingSpace value
```

EXAMPLE

To reduce intra-layer capacitance on critical nets, enter

set droute_timingSpace 1

droute_treatTiedFillAsFillToFillSpacing

DESCRIPTION

The **droute_treatTiedFillAsFillToFillSpacing** variable specifies whether fill-to-fill spacing is applied to existing tied fill. If the value is set to 0 (the default), the router follows fill-to-route spacing for existing tied fill during metal fill insertion. If the value is set to 1, the router follows fill-to-fill spacing for existing tied fill.

This variable applies only to the classic router.

SYNTAX

```
set droute_treatTiedFillAsFillToFillSpacing value
```

EXAMPLE

To use fill2-fill spacing for existing tied fill during metal fill, enter

set droute_treatTiedFillAsFillToFillSpacing 1

droute_trimUserAntenna

DESCRIPTION

The **droute_trimUserAntenna** variable determines how user-created dangling wires are handled.This is useful for creating a long bus, and the router can trim the unused part after connecting pins to the bus. Note that antenna here means dangling wires.

When set to 0 (the default), user-created dangling wires are not deleted. When set to 1, user-created dangling wires, such as a prerouted bus, are trimmed after search and repair or route-net group.

This variable applies only to the classic router.

SYNTAX

```
set droute_trimUserAntenna value
```

EXAMPLE

To trim user-created dangling wires after search and repair, enter

set droute_trimUserAntenna 1

droute_ultraWideWireMode

DESCRIPTION

The **droute_ultraWideWireMode** variable specifies the routing grid used for ultra wide wire routing. If the value is set to 0 (the default), the regular routing grid is used. If the value is set to 1, a wide routing grid is used. If the value is set to a larger value (up to 4), even wider grids are used. Setting a large value for ultra wide wires can lead to fast routing time due to the coarser grid for them. This is useful for flip-chip designs.

This variable applies only to the classic router.

SYNTAX

```
set droute_ultraWideWireMode value
```

EXAMPLE

To use wide grids to route ultra-wide wires, enter

set droute_ultraWideWireMode 1

droute_wireWidenForceWrongWay

DESCRIPTION

The **droute_wireWidenForceWrongWay** variable specifies whether to force widened wrong-way wires. If the value is set to 0 (the default), the router does not force widened wrong-way wires. If the value is set to 1, the router forces widened wrong-way wires to improve the coverage of widened wires.

This variable applies only to the classic router.

SYNTAX

```
set droute_wireWidenForceWrongWay value
```

EXAMPLE

To force widening wrong way wires, enter

set droute_wireWidenForceWrongWay 1

droute_wireWidenIgnoreMinEdgeLengthVio

DESCRIPTION

The **droute_wireWidenIgnoreMinEdgeLengthVio** variable specifies whether to ignore the minimum edge length rule during wire widening. If the value is set to 0 (the default), the router checks all DRC violations during wire widening. If the value is set to 1, the router does not check minimum edge length violations during wire widening.

This variable applies only to the classic router.

SYNTAX

```
set droute_wireWidenIgnoreMinEdgeLengthVio value
```

EXAMPLE

To close minimum edge length DRC violation checking in try mode, enter

set droute_wireWidenIgnoreMinEdgeLengthVio 1

droute_wireWidenPieceWise

DESCRIPTION

The **droute_wireWidenPieceWise** variable specifies whether to widen a wire segment uniformly or to widen it piece by piece. If the value is set to 0 (the default), the router widens a wire segment uniformly. If the value is set to 1, the router first breaks a wire into several pieces according to its neighboring routing objects, and then tries to widen each piece with a different width. This nonuniform widening can improve the coverage of widened wires.

This variable applies only to the classic router.

SYNTAX

```
set droute_wireWidenPieceWise value
```

EXAMPLE

To break a wire according to its neighboring routing objects and then widen each segment with different width, enter

set droute_wireWidenPieceWise 1

droute_wireWidenSrLoop

DESCRIPTION

The **droute_wireWidenSrLoop** variable specifies the number of search and repair loops after wire widening. If the value is set to N, the router runs N loops of search and repair after wire widening to fix DRC violations created during wire widening.

This variable applies only to the classic router.

SYNTAX

```
      set droute_wireWidenSrLoop value

The valid values of this variable range between 0 and 100.
The default is 10.
```

EXAMPLE

To specify 20 search and repair loops after wire widening, enter

set droute_wireWidenSrLoop 20

droute_wireWidenTimingDriven

DESCRIPTION

The **droute_wireWidenTimingDriven** variable specifies whether to widen wires on timing-critical nets. If the value is set to 0 (the default), the router does wire widening on all signal wires. If the value is set to 1, the router does not widen wires on timing-critical nets.

This variable applies only to the classic router.

SYNTAX

```
set droute_wireWidenTimingDriven value
```

EXAMPLE

To enable the router to freeze timing-critical nets while widening wires, enter

set droute_wireWidenTimingDriven 1

droute_wireWidenWidthScheme

DESCRIPTION

The **droute_wireWidenWidthScheme** variable specifies how many different widths are used for wire widening. If the value is set to N, the router tries N different widths during wire widening.

This variable applies only to the classic router.

SYNTAX

```
      set droute_wireWidenWidthScheme value

The valid values of this variable ranges between 1 and 5.
The default is 2.
```

EXAMPLE

To specify four different widths during wire widening, enter

set droute_wireWidenWidthScheme 4

droute_wrongWayExtraCost

DESCRIPTION

The **droute_wrongWayExtraCost** variable specifies extra wrong-way wire cost. The default wrong-way unit distance cost is 2 (versus default right-way unit distance cost of 1).If the value is set to 0, no extra wrong-way cost is added. Otherwise, if the value is set to N, the wrong-way unit distance cost is 2 + N. Therefore, if the value is set to 2, the wrong-way cost is four times the right-way unit distance cost.This variable can be used to further discourage wrong-way direction routing. It can be especially useful if you observe excessive wrong-way direction routing during the **route net group** command, where most nets are not routed yet.

This variable applies only to the classic router.

SYNTAX

```
     set droute_wrongWayExtraCost value

The valid values of this variable range between 0 and 100.
The default is 0.
```

EXAMPLE

To specify extra wrong-way wire cost as 40, enter

set droute_wrongWayExtraCost 40

duplicate_ports

Specifies whether ports are to be drawn on every sheet for which an input or output signal appears.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

This variable controls the partitioning option that specifies if ports are to be drawn on every sheet for which an input or output signal appears. When set to **true**, no off-sheet connectors are used for input and output signals, and signal ports are duplicated where indicated on each sheet.

To determine the current value of this variable, use the **printvar duplicate_ports** command. For a list of all schematic variables and their current values, use the **print_variable_group schematic** command.

echo_include_commands

Controls whether the contents of a script file are printed as it executes.

TYPE

Boolean

DEFAULT

true

GROUP

system_variables

DESCRIPTION

When this variable is set to **true** (the default value), the **include** command prints the contents of files as it executes. When set to **false**, the contents of the files are not printed.

To determine the current value of this variable, use the **printvar echo_include_commands** command. For a list of all system variables and their current values, use the **print_variable_group system** command.

eco_netlist_ignore_tie_changes

Specifies that the **eco_netlist** command should ignore connection changes from a tie to another tie on a pin or a port.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

If the **eco_netlist_ignore_tie_changes** variable is set to **true**, the **eco_netlist** command ignores connection change from a tie to another tie on a pin or a port.

The following connection changes are ignored:

- A direct tie connected to a pin or port changes to an indirect tie.
- An indirect tie connected to a pin or port changed to a direct tie.
- An indirect tie connected to a pin or port changed to another indirect tie.

The original connection and the new connection should be both tie high or tie low. Otherwise, the connection change is applied.

SEE ALSO

`eco_netlist(2)`

eco_netlist_preprocess_for_verilog

Enables the **eco_netlist -by_verilog_file** command to preprocess the input Verilog file.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Enables the **eco_netlist -by_verilog_file** command to preprocess the input Verilog file before comparing the netlists.

The preprocessing helps to minimize the following two kinds of unnecessary differences in the golden Verilog:

- Dangling SYNOPSYS_UNCONNECTED_* nets of open bus pins
  When bus bits of a vector pin are unconnected in the design database, the Verilog writer creates and connects a SYNOPSYS_UNCONNECTED_* net for each bit to output the vector as a bus. For example, assume port A is a bus in range of [3:0]. A[3] is connected to net n1, A[0] to n2, and A[1] and A[2] are open. The file generated by the **write_verilog** command is

```
sub ci ( .A( { n1, SYNOPSYS_UNCONNECTED_1, SYNOPSYS_UNCONNECTED_2, n2 } )
);
```

- These nets are treated as regular nets by the Verilog reader. Because they are actually not in database, the **eco_netlist** command assumes they are new nets and writes out the following changes:

```
create_net SYNOPSYS_UNCONNECTED_1
create_net SYNOPSYS_UNCONNECTED_2
connect_net SYNOPSYS_UNCONNECTED_1 ci/A[2]
connect_net SYNOPSYS_UNCONNECTED_2_ci/A[1]
```

- Dangling tie-low connections of hierarchical ports
  Sometimes a hierarchical port does not have a tie-low connection in the design database. But the Verilog reader always creates a same-name tie-low net for each dangling port. The **eco_netlist** command assumes that these are new nets and writes out the following changes:

```
create_net A
connect_net A A
```

If this variable is set to **true**, the **eco_netlist** command suppresses these two kinds of unnecessary differences.

SEE ALSO

`eco_netlist(2)`

eco_record_cell_change

Controls whether the **eco_netlist** command creates cell collections for the **create_cell**, **insert_buffer**, **change_link**, and **size_cell** commands in the change list. This variable has an effect only when you use the **-by_verilog_file** or **-by_tcl_file** option with the **eco_netlist** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Controls whether the **eco_netlist** command creates cell collections for the **create_cell**, **insert_buffer**, **change_link**, and **size_cell** commands in the change list. This variable has an effect only when you use the **-by_verilog_file** or **-by_tcl_file** option with the **eco_netlist** command.

By default (false), the **eco_netlist** command does not create cell collections.

When you set this variable to **true**, the command creates the following four cell collections:

- eco_netlist_created_cells (for the create_cell command)
- eco_netlist_inserted_buffers (for the insert_buffer command)
- eco_netlist_changed_links (for the change_link command)
- eco_netlist_sized_cells (for the size_cell command)

These collections are runtime data and are not stored in the Milkyway database. They are valid only until the Milkyway cell is closed.

You can use the **place_eco_cells** command with these collections to handle them separately.

SEE ALSO

```
eco_netlist(2)
```

enable_bit_blasted_bus_linking

Resolves the mismatches between bit-blasted bus bits and the buses during the link process.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

Setting this variable to true allows the linker to recognize bit-blasted buses by pin names during the link process. If the pin names follow a specific pattern, the tool infers a bus when encountering the individual bits.

Typically, the RTL pin names and the logic library pin names should match. Signals are defined the same way in both the RTL and the library. They are defined as buses or a bus is defined by its individual wires. Occasionally, mismatches occur during design development; for example, when you transfer the design from one technology to another. To fix the mismatches, you can set the **enable_bit_blasted_bus_linking** variable to true before the design is read.

When you set the **enable_bit_blasted_bus_linking** variable to true, the linker allows you to read your design even if pin name mismatches occur. By default, this variable is set to false, but the default for DC Explorer is true. Note that when this variable is set to true, the tool matches pin names according to the setting of the **bit_blasted_bus_linking_naming_styles** variable.

You specify the pin name patterns by using the **bit_blasted_bus_linking_naming_styles** variable. For example:

```
prompt> set bit_blasted_bus_linking_naming_styles {%s[%d]}
```

This permits the U1 reference

```
input [1:0] in;
my_macro U1( .S(in),...
```

to link even if my_macro is defined as follows:

```
my_macro
input S[1];
input S[0];
\...
```

To determine the current value of this variable, use the **printvar enable_bit_blasted_bus_linking** command. For a list of all system variables and their current values, use **print_variable_group system**.

SEE ALSO

`bit_blasted_bus_linking_naming_styles(3)`

enable_cell_based_verilog_reader

Turns on the verilog2cel Verilog reader.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

This variable enables the verilog2cel Verilog reader. which is a CEL-based Verilog reader that saves the netlist directly into the Milkyway CEL, without creating tool data structures.

To determine the current value of this variable, use the **printvar enable_cell_based_verilog_reader** command. For a list of all HDL variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

```
read_verilog(2)
mw_current_design(3)
```

enable_clock_to_data_analysis

Enables the usage of latency and transition time of ideal clocks for timing analysis in clock-to-data signal paths.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

This variable, when set to **true**, enables timing analysis of clock-to-data pins when the clock is ideal. The ideal clock latency is used as the data arrival time at the clock-to-data pin, and the ideal transition is used as data transition time at that pin. If more than one clock drives the data pin, the worst latency and transition values apply.

When the variable is set to **false** (the default), the tool uses the actual propagated clock latency and transition time at the clock-to-data pin, if any, and ignores any ideal clock latency and transition time set for the clock.

A clock-to-data pin is a cell's data input pin in the fanout of a clock, such as the input of a clock divider circuit. A pin attribute called pin_is_clock_to_data is set to true for these pins.

You can set the ideal latency and ideal transition time of a clock by using the **set_clock_latency** and **set_clock_transition** commands.

SEE ALSO

```
create_clock(2)
create_generated_clock(2)
set_clock_latency(2)
set_clock_transition(2)
```

enable_golden_upf

Enables the golden UPF mode when set to true.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable, when set to true, enables the golden UPF mode. By default, the variable is set to false and the golden UPF mode is disabled. In that case, the **save_upf** command writes out a full set of UPF commands that reflect the changes made by the tool to the UPF power intent of the design, as well as the UPF commands run during the tool session.

In the golden UPF mode, the same original "golden" UPF script file is used throughout the synthesis, physical implementation, and verification flow. The **save_upf** command writes out a supplemental UPF file that reflects only the changes made by the tool to the UPF power intent of the design. Downstream tools and verification tools use the golden and supplemental UPF files together to completely specify the power intent of the design. To use the golden UPF mode, set this variable to true before you execute any UPF commands. Once enabled, the golden UPF mode requires that you run UPF commands only by using the **load_upf** command. You cannot execute individual UPF commands at the shell prompt, except for UPF query commands.

In the golden UPF mode, the **save_upf** command writes out a supplemental UPF file, which contains only the UPF changes made by the tool during the session, such as tool-derived power intent and addition of power management cells. You can choose to write out a concise or verbose supplemental UPF file. A verbose file contains **connect_supply_net** commands that connect supply nets to the power management cells; these commands are needed if the netlist written in the session does not include PG connections. If the netlist includes PG connections, you can write out a concise supplemental UPF file instead.

To determine the current value of this variable, use **printvar enable_golden_upf** command. For a list of all mv variables and their current values, use the **print_variable_group mv** command.

SEE ALSO

```
load_upf(2)
save_upf(2)
upf_name_map(3)
```

enable_instances_in_report_net

Enables the **report_net** command to report on instances in the current design.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

This variable enables the **report_net** command to report on instances in the current design, when set to **true** (the default value).

When this variable is set to **false**, the **report_net** command reports only nets for the current design. Also, nets in the current instance are reported if the **current_instance** command is set.

SEE ALSO

`report_net(2)`

enable_page_mode

Controls whether long reports are displayed one page at a time (similar to the UNIX **more** command).

TYPE

Boolean

DEFAULT

true

GROUP

system_variables

DESCRIPTION

This variable, when set to **true**, displays long reports one page at a time (similar to the UNIX **more** command). Commands affected by this variable include **list**, **help**, and the **report** commands.

To determine the current value of this variable, use the **list enable_page_mode** command. For a list of all system variables and their current values, use the **print_variable_group system** command.

enable_recovery_removal_arcs

Controls whether the tool accepts recovery and removal arcs specified in the technology library.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

This variable, when set to **true**, enables the acceptance of recovery and removal arcs specified in the technology library. Recovery or removal timing arcs impose constraints on asynchronous pins of sequential cells. Typically, recovery time specifies the time the inactive edge of the asynchronous signal has to arrive before the closing edge of the clock. Removal time specifies the length of time the active phase of the asynchronous signal must be held after the closing edge of clock.

To enable the **compile**, **report_timing**, and **report_constraint** commands to accept recovery or removal arcs specified in the library, set **enable_recovery_removal_arcs** to **true**.

Note that independent of the value of this variable, the **write_timing** and **report_delay_calculation** commands always accept and report recovery or removal timing information.

This variable is the logical opposite of the variable **timing_disable_recovery_removal_checks**. If you set either one of these variables to **true**, the tool automatically sets the other variable to **false**, and vice versa.

SEE ALSO

`timing_disable_recovery_removal_checks(3)`

enable_slew_degradation

Determines whether the transition degradation is considered for nets with physical information.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

When this variable is set to **true** (the default value), timing calculation takes into account transition degradation across a net. For a long net, this might mean that the transition at the net driver could be very different than the various load pins. Because all load pins on the same net might not have the same transition time, optimization with physical information fixes transitions on a per pin basis. This might lead to increased runtime for these commands.

Setting this variable to **false** causes the timing calculation to not calculate transition degradation across a net. However, some libraries do have a transition degradation table that shows how to degrade the transition across a net. If slew degradation is not enabled, then the library transition degradation tables are used unless the **disable_library_transition_degradation** variable is set to **true**.

If neither slew degradation nor library transition degradation is enabled, the transition at the net driver will be the same transition at the load pin. Because all load pins on the same net have the same transition time, optimization with physical information attempts to fix transitions on a per net basis.

Regardless of the setting of this variable, slew degradation does not occur for multidriven nets. For multidriven nets, there can be cases of too much pessimism.

Regardless of the setting of this variable, slew degradation can only occur if there is either delay back-annotation for the net or physical locations for the pins of the net. Otherwise, the transition time calculation at the driver pin is too inaccurate to be degraded.

To determine the current value of this variable, use the **printvar enable_slew_degradation** command. For a list of all timing variables and their current values, use the **print_variable_group timing** command.

SEE ALSO

```
disable_library_transition_degradation(3)
timing_variables(3)
```

enable_special_level_shifter_naming

Enables special naming for automatically-inserted level shifters.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

When this variable is set to **true**, automatically-inserted level shifters by the **insert_level_shifters**, **compile**, and other commands are specially named. The name uses the following template:

<*prefix*> + <*PD OR Design name*> + "_LS" + #

The <*prefix*> is a user-specified prefix specified by the **level_shifter_naming_prefix** variable. The # is a number internally generated to keep this name unique. The <*PD OR Design Name*> is the name of the power domain where the level shifter is being added, or the design name if the power domain is not defined.

SEE ALSO

`level_shifter_naming_prefix(3)`

estimate_io_latency

Uses estimated I/O latency in timing calculations for ports when set to **true**.

TYPE

Boolean

DEFAULT

false

GROUP

timing

DESCRIPTION

This variable uses estimated I/O latency in timing calculations for ports when the value is set to **true**.

For each input port the I/O latency for max case is defined as the minimum clock network latency for the clocks in the fanout set of the port. The I/O latency for min case is the maximum clock latency in the fanout set.

For each output port the I/O latency for max case is defined as the maximum clock network latency for the clocks in the fanin set of the port. The I/O latency for min case is the minimum clock latency in the fanin set.

The I/O latency is not added to external delay in timing calculations if the **network_delay_included** switch is used. The ideal clocks are not used for I/O latency calculation.

This variable is best used after clock tree synthesis.

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

SEE ALSO

```
report_timing(2)
set_clock_latency(2)
set_propagated_clock(2)
```

exit_delete_command_log_file

Controls whether the file specified by the **command_log_file** variable is deleted after **design_analyzer** or dc_shell exits normally.

TYPE

string

DEFAULT

false

GROUP

system_variables

DESCRIPTION

When this variable is set to **true**, the file specified by the **command_log_file** variable is deleted after **design_analyzer** or dc_shell exits normally. The default value is **false**. Set **exit_delete_command_log_file** to **false** to retain the file.

To determine the current value of this variable, use the **printvar exit_delete_command_log_file** command. For a list of all system variables and their current values, use **print_variable_group system**.

SEE ALSO

`command_log_file(3)`

exit_delete_filename_log_file

Controls whether the file specified by the **filename_log_file** variable is deleted after **design_analyzer** or dc_shell exits normally.

TYPE

string

DEFAULT

true

GROUP

system_variables

DESCRIPTION

When this variable is set to **true** (the default value), the file specified by the **filename_log_file** variable is deleted after **design_analyzer** or dc_shell exits normally. Set **exit_delete_filename_log_file** to **false** to retain the file.

To determine the current value of this variable, use the **printvar exit_delete_filename_log_file** command. For a list of all system variables and their current values, use **print_variable_group system**.

SEE ALSO

`filename_log_file(3)`

extract_max_parallel_computations

Sets the maximum degree of parallelism in multi-core extraction.

TYPE

Integer

DEFAULT

0

GROUP

physopt

DESCRIPTION

This application variable specifies the degree of parallelism in multi-core extraction. Increasing parallelism results in reduced runtime.

When set to 0, the number of cores used for extraction is specified directly by the value of the set_host_options command with the -max_cores option value.

When set to 1, multicore extraction is disabled.

When the value is smaller than the -max_cores value of the set_host_options command, the number of cores being used will not exceed this limit.

```
prompt> set extract_max_parallel_computations 4
prompt> extract_rc

EKL_MT: max_num_of_threads = 4
EKL_MT: total threadable CPU 1.74(= 0.44 + 0.44 + 0.43 + 0.43) seconds
EKL_MT: elapsed time 0 seconds
```

SEE ALSO

set_host_options(2)

```
extract_rc(2)
```

fanin_fanout_trace_arcs

Specifies the type of combinational arcs to trace during **all_fanin** and **all_fanout**.

TYPE

string

DESCRIPTION

This variable specifies the type of combinational arcs to trace during **all_fanin** and **all_fanout** commands, when that command's **-trace_arcs** option is not used.

Allowed values are **timing**, which permits tracing only of valid timing arcs (that is, arcs which are neither disabled nor invalid due to case analysis); and **all**, which permits tracing of all combinational arcs regardless of either case analysis or arc disabling.

The default in DC/ICC is timing; the default in DC Explorer is all.

If the all_fanin/all_fanout command specifies option -trace_arcs, then that option takes effect regardless of the value of this variable.

SEE ALSO

```
all_fanin(2)
all_fanout(2)
```

filename_log_file

Specifies the name of the filename log file to be used in case a fatal error occurs during execution of **design_analyzer** or dc_shell.

TYPE

string

DEFAULT

filenames.log

GROUP

system_variables

DESCRIPTION

This variable specifies the name of the filename log file to be used in case a fatal error occurs during execution of **design_analyzer** or dc_shell. The file specified by **filename_log_file** contains all filenames read in by **design_analyzer** or dc_shell, including .db, script, Verilog, VHDL, and include files for one invocation of the program. If there is a fatal error, you can easily identify the data files needed to reproduce the fatal error. If this variable is not specified, the default filename **filenames.log** is used. This file is deleted if the program exits normally, unless **exit_delete_filename_log_file** is set to **false**.

If the application has been invoked using the **-no_log** switch, then the process ID of the application and the timestamp is appended to the filename log file in the following format:

filenames_<*process_id*>_<*timestamp*>.log.

To determine the current value of this variable, use the **printvar filename_log_file** command. For a list of all system variables and their current values, use **print_variable_group system**.

SEE ALSO

`exit_delete_filename_log_file(3)`

find_allow_only_non_hier_ports

Instructs the **find** command to search for ports in subdesigns.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

When this variable is set to **true**, the **find** command searches for top-level ports only, ignoring ports in subdesigns.

By default, the **find port** command will also fetch ports in the subdesigns.

For example, if cell *c* is an instance of design *SUB1*, and this variable is set to **false** the command **find port c/A** fetches the port *A* on design *SUB1*. With the variable set to **true**, the command **find port c/A** results in a warning message as shown below:

```
prompt> printvar find_allow_only_non_hier_ports
find_allow_only_non_hier_ports = "false"

prompt> find port c/A
{"c/A"}

prompt> find_allow_only_non_hier_ports = true
"true"

prompt> printvar find_allow_only_non_hier_ports
find_allow_only_non_hier_ports = "true"

prompt> find port c/A
Warning: Can't find port 'c/A' in design 'SUB1'. (UID-95)
```

This variable is used to facilitate the netlist editing commands **connect_net** and **disconnect_net** which allow net and pin instances in addition to handling nets and pins in the current design.

SEE ALSO

```
connect_net(2)
disconnect_net(2)
get_ports(2)
remove_port(2)
```

find_converts_name_lists

Controls whether the **find** command converts the *name_list* string to a list of strings before searching for design objects.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

When this variable is set to **true**, the **find** command converts the *name_list* string to a list of strings before searching for design objects. In addition, when this variable is **true**, all commands that use the implicit **find** will convert appropriate strings to lists of strings before searching for objects. For example, **current_instance** uses the implicit **find** command and would convert the string *instance* to a list of strings before searching for objects.

The **find_converts_name_lists** variable provides backward compatibility with the premodification **find** command. When the variable is **false** (the default value), **find** executes according to its postmodification behavior; that is, strings are not converted to lists of strings.

To determine the current value of this variable, use the **printvar find_converts_name_lists** command. For a list of all system variables and their current values, use **print_variable_group system**.

SEE ALSO

```
current_instance(2)
system_variables(3)
```

find_ignore_case

Controls whether the **find** command is case-sensitive when matching object names.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Normally, the **find** command is case-sensitive when matching names of objects. That is, for either an explicit or implicit invocation of the **find** command, **find** only matches objects whose names have the same case as the specified string.

When this variable is set to **true**, the **find** command performs case-insensitive string comparisons. This variable affects both implicit and explicit invocations of the **find** command.

To determine the current value of this variable, use the **list find_ignore_case** command. If the variable has not yet been defined, the **list** command will fail, indicating that the variable is undefined.

Use this variable only when necessary, as it may slightly increase the runtime of **find** operations.

SEE ALSO

floorplan_data_attributes

Contains attributes related to floorplan data on different modules in separate views.

DESCRIPTION

Contains attributes related to floorplan data on different modules in separate views.

Floorplan data attributes are read-only. You can use **get_floorplan_data** to determine value of an attribute, and use **report_floorplan_data** to get a report of all the floorplan data on module objects in a specified view. If you want to know the list of valid floorplan data names, you can use **list_floorplan_data**.

Floorplan Data Attributes

area

Specifies the area of a module. This attribute does not exist on a logical module.

The tool calculates the attribute based on cell boundary.

aspect_ratio

Specifies the **height**:**width** ratio of a module. This attribute does not exist on a logical module.

hard_macro_area_percentage

Specifies ratio of sum of **area** of hard macros in a module to top module's **physical_area**.

height

Specifies the height of a module. This attribute only exists on soft macros or hard macros in **logical** view.

The tool calculates the attribute based on the **bbox** of a module.

macro_area_percentage

Specifies the ratio of sum of **area** of macros in a module to top module's **physical_area**.

Note that the definition of **macro_area_percentage** in this document is different from that in **cell_attributes.3**. Compared to the definition in **cell_attributes.3**, the numerator of ratio remains while the denominator is **physical_area** of top module instead of **physical_area** of the specified module.

number_of_black_box

Specifies the count of black boxes in a module. This attribute does not exist on plan_group.

Whether a cell is a black box can be determined by the attribute **is_black_box**.

number_of_hard_macro

Specifies the count of hard macros in a module.

Whether a cell is a hard macro can be determined by the attribute **is_hard_macro**.

number_of_io_cell

Specifies the count of io cells in a module. This attribute does not exist on **plan_group**.

The tool counts cells in when its **mask_layout_type** is **io_pad**, **corner_pad**, **pad_filler**, or **flip_chip_pad**.

number_of_macro

Specifies the count of macros in a module. This attribute does not exist on **plan_group**.

The tool counts cells in when its **mask_layout_type** is **macro**.

number_of_standard_cell

Specifies the count of standard cells in a module.

The tool counts cells in when its **mask_layout_type** matches ***std***.

physical_area

Specifies the physical area of a module.

The physical area of a hard macro is equal to its **area**.

In **one_level** or **logical** view, the physical area of a soft macro is equal to its **area**. However, in **physical** view, physical area is the sum of **physical_area** of macros and **area** of standard cells inside the specified soft macro.

The physical area of a plan group is sum of **area** of macros and standard cells in it.

For a top module or a logical module, rules for calculating **physical_area** are applied as:

In a **one_level** view, only children in the current level are counted and children should be either macro or standard cell. The physical area of a module in **one_level** is sum of the **area** of the counted children.

In a **logical** view, the hierarchy tree in the module is traversed and the **area** of macros and standard cells in the tree will be added.

In the **physical** view, the hierarchy tree in the module is traversed and the **area** of standard cells and **physical_area** of macros in the tree will be added.

physical_area_percentage_in_top_design

Specifies the ratio of **physical_area** of a module to top module's **physical_area**.

utilization

Specifies the utilization of a module.

The tool calculates the attribute using **physical_area**:**area** ratio of a module. This attribute does not exist on a logical module or a top module.

width

Specifies the width of a module. This attribute only exists on soft macros or hard macros in **logical** view.

The tool calculates the attribute based on the **bbox** of a module.

SEE ALSO

```
get_floorplan_data(2)
list_floorplan_data(2)
report_floorplan_data(2)
```

focalopt_endpoint_margin

Read and set the endpoint slack value during focal_opt setup or hold optimization.

TYPE

Boolean

DEFAULT

true

GROUP

focalopt_variables

DESCRIPTION

It enables the endpoint margin feature for focal_opt. This features only applies to the **-setup_endpoints** and **-hold_endpoints** options of focal_opt.

When variable is set to true, the tool will read the slack value for the respective endpoint specified in the setup_endpoint file or hold_endpoint file and overwrite the tool's current slack value with the specified value.

When variable is set to false, the tool will only optimize the endpoint according to the tool's current slack value.

This feature is useful for tackling correlation issues for specific endpoints in postroute optimization stage.

Please refer to **focal_opt(2)** man page for accepted FILE formats.

SEE ALSO

`focal_opt(2)`

focalopt_power_critical_range

Specifies the slack threshold for performing final stage leakage-power recovery with the **focal_opt -power** command.

TYPE

float

DEFAULT

0

DESCRIPTION

This variable specifies the power critical range in design timing units. The power critical range sets the slack threshold for performing final stage leakage-power recovery. You can specify a positive number, a negative number, or zero.

If the worst slack of all the paths through a cell is greater than the specified value, the **focal_opt -power** command performs final stage leakage-power recovery for that cell. Final stage leakage-power recovery preserves the timing QoR such that the resulting slack is greater than or equal to the specified value.

If a path has a slack value less than the specified value, none of the cells in that path are optimized.

By default, the power critical range value is zero, and leakage-power recovery is performed only those cells with positive slack (cells on nonviolating paths).

When you set this variable to a positive number, the command preserves timing QoR with positive slack and performs less leakage-power recovery. When you set this variable to a negative number, the command performs more aggressive leakage-power recovery and allows timing degradation.

This variable applies only to the **focal_opt -power** command. This variable applies to all scenarios in the design.

EXAMPLE

The following example defines the power critical range to be 10. If the timing unit for the design is picoseconds, this value means 10ps. The **focal_opt -power** command performs

leakage-power recovery on cells with a worst slack greater than 10ps and maintains a worst slack of at least 10ps.

```
prompt> set_app_var focalopt_power_critical_range 10
```

The following example defines the power critical range to be -10. If the timing unit for the design is picoseconds, this value means -10ps. The **focal_opt -power** command performs leakage-power recovery on cells with a worst slack greater than -10ps and can degrade the worst slack up to -10ps.

```
prompt> set_app_var focalopt_power_critical_range -10
```

SEE ALSO

`focal_opt(2)`

fp_allocate_mcmm_script_dir

Specifies the name of the directory to save the scripts generated during budgeting of a multicorner-multimode design.

TYPE

string

DEFAULT

fp_mcmm_scripts

GROUP

budgeting_variables

DESCRIPTION

When budgeting a multicorner-multimode design, separate sdc constraint files are generated for each active scenario. Budgeting also generates a script for each block that can be used to load these constraints. Use this variable to specify the name of the directory where all the scripts are saved.

SEE ALSO

`allocate_fp_budgets(2)`

fp_bb_flow

Enables complete timing operations on designs containing black boxes in the early design planning stage.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable allows the completion of timing operations on designs that contain black boxes without frame views. Use this variable in the early design planning stages only. In the implementation stage, black boxes should have both timing models and frame views for greater accuracy.

SEE ALSO

`allocate_fp_budgets(2)`

fpopt_no_legalize

Used to disable legalization step during optimize_fp_timing.

TYPE

Boolean

DEFAULT

false

GROUP

optimize_fp_timing_variables

DESCRIPTION

Optimize_fp_timing involves embedded legalization step. This variable can be used to disable the legalization step.

SEE ALSO

`optimize_fp_timing(2)`

fsm_auto_inferring

Determines whether or not to automatically extract the finite state machine during **compile**.

TYPE

Boolean

DEFAULT

false

GROUP

fsm_variables

DESCRIPTION

This variable determines whether the compiler performs finite state machine (FSM) automatic extraction during **compile**. If a design has previously had state machine extraction performed on it, the compiler does not perform extraction again with the same encoding style.

EXAMPLES

In the Presto flow, the compiler detects the FSM attributes from the HDL code and extracts the FSM during **compile**, as shown here:

```
prompt> fsm_auto_inferring=true
prompt> read -f verilog example.v
prompt> set_fsm_encoding_style one_hot
prompt> compile
```

SEE ALSO

```
fsm_enable_state_minimization(3)
fsm_export_formality_state_info(3)
```

fsm_enable_state_minimization

Determines whether or not the state minimization is performed for all finite state machines (FSMs) in the design.

TYPE

Boolean

DEFAULT

false

GROUP

fsm_variables

DESCRIPTION

When this variable is set to **true**, the compiler performs the state minimization after the state extraction.

SEE ALSO

```
fsm_auto_inferring(3)
fsm_export_formality_state_info(3)
```

fsm_export_formality_state_info

Determines whether or not state machine encoding information is exported into the files that will be used by Formality.

TYPE

Boolean

DEFAULT

false

GROUP

fsm_variables

DESCRIPTION

When this variable is set to **true**, the state encoding information before and after finite state machine extraction is exported into two files, <*module_name*>.ref and <*module_name*>.imp, which will be used for Formality verification. The default value for this variable is **false**.

SEE ALSO

```
fsm_auto_inferring(3)
fsm_enable_state_minimization(3)
```

gen_bussing_exact_implicit

Controls whether schematics generated using the **create_schematic -implicit** command should contain implicit bus names instead of bus rippers.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

When this variable is set to **true**, it specifies that schematics generated using the **create_schematic -implicit** command should contain no bus rippers. Instead, all bused connections are to be shown with implicit bus names.

Use this variable with the **-implicit** command-line option. By default, schematics generated using the **-implicit** option have rippers between any bus connections where the ripper connects to a pin in a column adjacent to the originating bus pin. Bused connections between cell pins more than one column away from each other are always shown disconnected with the **-implicit** option.

To determine the current value of this variable, use the **printvar gen_bussing_exact_implicit** command. For a list of all schematic variables and their current values, use **print_variable_group schematic**.

SEE ALSO

gen_cell_pin_name_separator

Specifies the character used to separate cell names and pin names in the bus names generated by the **create_schematic** command.

TYPE

string

DEFAULT

/

GROUP

schematic_variables

DESCRIPTION

If this variable is set, then its value is used to separate the cell and pin names in the bus names generated by **create_schematic**. By default, a / (slash) is used to separate the cell and pin names, thus creating bus names such as U0/OUT[0:3].

To determine the current value of this variable, use the **printvar gen_cell_pin_name_separator** command. For a list of all schematic variables and their current values, use **print_variable_group schematic**.

SEE ALSO

gen_create_netlist_busses

Controls whether **create_schematic** creates netlist buses whenever it creates buses on the schematic.

TYPE

Boolean

DEFAULT

true

GROUP

schematic_variables

DESCRIPTION

Controls whether the **create_schematic** command creates netlist buses whenever it creates buses on the schematic. When the value is **true** (the default), the **create_schematic** command creates netlist buses. Usually this occurs when **create_schematic** creates buses to connect to bused pins on the schematic. But it might also occur when there are bused ports on the schematic.

To determine the current value of this variable, use the **printvar gen_create_netlist_busses** command. For a list of all **schematic** variables and their current values, use **print_variable_group schematic**.

SEE ALSO

gen_dont_show_single_bit_busses

Controls whether single-bit buses are generated in the schematic.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

This variable is used in conjunction with the **gen_show_created_busses** variable. When **gen_show_created_busses** is set to **true** and **gen_dont_show_single_bit_busses** is also set to **true**, single-bit buses created by gen are not printed out. This suppresses messages about the creation of single-bit buses in the schematic.

To determine the current value of this variable, use the **printvar gen_dont_show_single_bit_busses** command. For a list of all schematic variables and their current values, use the **print_variable_group schematic** command.

SEE ALSO

gen_match_ripper_wire_widths

Controls whether the **create_schematic** command generates rippers whose width always equals the width of the ripped net.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

This variable controls whether the **create_schematic** command generates rippers whose width always equals the width of the ripped net. When set to **true**, any rippers whose wire ends connect to scalar nets are of unit width.

By default, the **create_schematic** command connects scalar nets to the wire ends of multibit rippers. In this case, all of the concerned bits of the ripper are assumed to be shorted together and connected to that scalar net.

To determine the current value of this variable, use the **printvar gen_match_ripper_wire_widths** command. For a list of all **schematic** variables and their current values, use **print_variable_group schematic**.

SEE ALSO

gen_max_compound_name_length

Controls the maximum length of compound names of bus bundles for the **create_schematic -sge** command.

TYPE

integer

DEFAULT

256

GROUP

schematic_variables

DESCRIPTION

This variable controls the maximum length of compound names of bus bundles for the **create_schematic -sge** command. The default length is 256, which is the maximum length supported by SGE. Any buses with names longer than this variable are decomposed into their individual members in the schematic. If any such buses connect to cells referencing library symbols, those library symbols are ignored and new gen-created symbols are used. Any buses that these gen-created symbols have whose compound names exceed this length are blown up into their individual members.

To determine the current value of this variable, use the **printvar gen_max_compound_name_length** command. For a list of all schematic variables and their current values, use the **print_variable_group schematic** command.

SEE ALSO

gen_max_ports_on_symbol_side

Specifies the maximum allowed size of a symbol created by the **create_schematic** command.

TYPE

integer

DEFAULT

0

GROUP

schematic_variables

DESCRIPTION

This variable specifies the maximum allowed size of a symbol created by the **create_schematic** command. For example, if this variable is set to 5, symbols with no more than 5 ports on any 1 side are created. If this variable is not set or is set to 0, all input ports are placed on the left side of the symbol and all inout and output ports are placed on the right side.

To determine the current value of this variable, use the **printvar gen_max_ports_on_symbol_side** command. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

gen_open_name_postfix

Specifies the postfix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins.

TYPE

Boolean

DEFAULT

""

GROUP

schematic_variables

DESCRIPTION

Specifies the postfix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins. The default is "".

The format of the net names is "%s%d%s", where the first "%s" is replaced by the value of **gen_open_name_prefix**, the second "%s" is replaced by the value of **gen_open_name_postfix**, and the "%d" is replaced by an integer whose value is generated automatically by **create_schematic -sge**.

For example, if **gen_open_name_prefix** = "Open", and **gen_open_name_postfix** = "_net", then the names created by **create_schematic** would be "Open1_net", "Open2_net", and so on.

The default values for **gen_open_name_prefix** and for **gen_open_name_postfix** are "Open" and "", respectively, so the default names created by **create_schematic** are Open1", "Open2", and so on.

To determine the current value of this variable, type **printvar gen_open_name_postfix**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

`gen_open_name_prefix(3)`

gen_open_name_prefix

Specifies the prefix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins.

TYPE

string

DEFAULT

Open

GROUP

schematic_variables

DESCRIPTION

Specifies the prefix to be used by **create_schematic -sge** when creating placeholder net names for unconnected pins. The default is "Open".

The format of the net names is "%s%d%s", where the first "%s" is replaced by the value of **gen_open_name_prefix**, the second "%s" is replaced by the value of **gen_open_name_postfix**, and the "%d" is replaced by an integer whose value is generated automatically by **create_schematic -sge**.

For example, if **gen_open_name_prefix** = "Open", and **gen_open_name_postfix** = "_net", then the names created by **create_schematic** would be "Open1_net", "Open2_net", and so on.

The default values for **gen_open_name_prefix** and for **gen_open_name_postfix** are "Open" and "", respectively, so the default names created by **create_schematic** are Open1", "Open2", and so on.

To determine the current value of this variable, type **printvar gen_open_name_prefix**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

`gen_open_name_postfix(3)`

gen_show_created_busses

Controls whether a message is printed out every time a schematic bus is created from cell pins for which no equivalent net bus exists in the netlist.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

When true, a message is printed out every time a schematic bus is created from cell pins for which no equivalent net bus exists in the netlist. The default is false.

To determine the current value of this variable, type **printvar gen_show_created_busses**. For a list of all **schematic** variables and their current values, type **print_variable_group schematic**.

SEE ALSO

gen_show_created_symbols

Controls whether **create_schematic** prints a warning message every time it generates a new symbol for a cell because an appropriate symbol could not be found in the symbol libraries.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

Controls whether **create_schematic** prints a warning message every time it generates a new symbol for a cell because an appropriate symbol could not be found in the symbol libraries. The default is false.

To determine the current value of this variable, use **printvar gen_show_created_symbols**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

SEE ALSO

gen_single_osc_per_name

Controls whether more than one off-sheet connector with any particular name is drawn on any schematic sheet.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

When this variable is set to true, only one off-sheet connector with any particular name is drawn on any schematic sheet. In cases where there could potentially be more than one off-sheet connector with the same name on any schematic page, only one connector is drawn and the others just have their net segments show up as unconnected stubs.

To determine the current value of this variable use **printvar gen_single_osc_per_name**. For a list of all **schematic** variables and their current values, use the **print_variable_group schematic** command.

SEE ALSO

generate_via_region_when_pin_width_all_less_than_via_width

TYPE

Boolean

DEFAULT

false

GROUP

extract_blockage_pin_via_variables

DESCRIPTION

This variable, when set to **true**, causes the **extract_blockage_pin_via** command to always generate via regions for pins, even when the width of the whole pin in smaller than the via width.

By default, this variable is set to **false**. In that case, at 65 nm technology nodes and below, where a pin is smaller than the minimum via size, no via region is generated.

Via Regions

In a FRAM view, a via region is an area consisting of one or more rectangular boxes, which may be overlapping, over a pin where the router can make a connection to a port. The via region provides guidance to the router about where to drop a via to make a connection.

By default, the router can use either a wire in the pin layer or a via above the pin layer to make a connection to a pin. In the IC Compiler tool, you can restrict the allowed methods of making connections in each layer by using the following command:

```
prompt> set_route_zrt_common_options \
          -connect_within_pins_by_layer_name \
          { {layer mode} {layer mode} ...}
```

When this option is set for a pin layer, the router attempts to make the lower-metal enclosure fall entirely within the metal pin in the FRAM view. This "within-pin" connection ensures that

the router will not introduce any DRC violations in the lower metal layer of the cell being connected.

However, if the router is unable to fit the lower-metal enclosure within the pin, it makes the connection anyway, causing the lower-metal enclosure to extend beyond the pin shape. This type of connection is said to "change the pin shape." In that case, the router must spend time checking for and fixing lower-metal design rule violations in the cell.

The manner of via region generation depends on the technology geometry size. For technology nodes at 90 nm and above, the blockage, pin, and via extraction process allows full flexibility to the router without considering whether the pin shape needs to be changed. For technology nodes at 65 nm and below, by default, the blockage, pin, and via extraction process creates more restrictive via areas that ensure "within-pin" via connections that do not change the metal pin shape, so that routing can be performed more quickly.

Enforcing all "within-pin" connections can be too restrictive in certain situations, such as when pin is small and the minimum enclosure around the via is large. In that case, you can override the default behavior of the **extract_blockage_pin_via** command by setting the **via_can_change_pin_shape** variable to **true**.

This allows the **extract_blockage_pin_via** command to create via regions that extend outside of pins for greater router flexibility, at the cost of more router runtime.

If pins are smaller than the minimum via size, then no via regions are generated at all by default. To allow via regions to be generated under these conditions, set the **generate_via_region_when_pin_width_all_less_than_via_width** variable to **true**. This causes the **extract_blockage_pin_via** command to generate via regions for pins, even when the width of the whole pin in smaller than the via width.

The IC Compiler tool lets you edit via regions in FRAM views so that you can control the behavior of the router in making connections to ports through vias. Use the commands **create_via_region**, **write_via_region** and **remove_via_region**.

SEE ALSO

```
extract_blockage_pin_via(2)
via_can_change_pin_shape(3)
```

generic_symbol_library

Specifies the generic symbol library used for schematics.

TYPE

string

DEFAULT

generic.sdb

GROUP

schematic_variables

DESCRIPTION

This variable specifies the .db file that contains generic symbols, templates, and layers used for schematics.

To determine the current value of this variable, use the **printvar generic_symbol_library** command. For a list of all schematic variables and their current values, use the **print_variable_group schematic** command.

golden_upf_report_missing_objects

Enables reporting of missing objects during Golden UPF reapplication.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

Setting this variable to true enables reporting of all missing objects when you reapply the golden UPF file to the design with the **load_upf** command.

In the golden UPF flow, the synthesis and physical implementation tools can change or remove objects in the design due to optimization, grouping, ungrouping, or expansion of wildcard names. Therefore, when you reapply the same golden UPF to the modified design, missing objects are expected as a normal part in the flow.

For this reason, by default, the tool does not report missing objects during reapplication of the golden UPF to the design, except for missing control signals identified by the **set_isolation_control**, **set_retention_control** and **create_power_switch** commands. These signals are not expected to be changed or removed by synthesis or physical implementation tools.

To enable reporting of all missing objects under these conditions, set the **golden_upf_report_missing_objects** to true. In that case, during reapplication of the golden UPF with the **load_upf** command, all missing objects are reported.

To determine the current value of this variable, use **printvar golden_upf_report_missing_objects** command. For a list of all mv variables and their current values, use the **print_variable_group mv** command.

SEE ALSO

```
load_upf(2)
save_upf(2)
enable_golden_upf(3)
upf_name_map(3)
```

ground_nets_for_upf_supply

Specifies a list of supply net names that identifies those nets as ground type supply nets in the IC Compiler multivoltage UPF flow.

TYPE

String

DEFAULT

""

GROUP

mv

DESCRIPTION

This variable, when set to a list of valid UPF supply net names, identifies those nets as ground type supply nets in the IC Compiler multivoltage UPF flow. This can be useful in a UPF black-box flow, when some supply nets are used in subblocks but are not used at the top level. By identifying these nets as ground nets, the IC Compiler tool has the information it needs to derive the newly created ground nets at the top level, and the ground type information is also honored by other multivoltage commands such as **check_mv_design** etc.

It is not necessary to use this variable to identify UPF supply nets that already have real usage in the design, for example, when they are used as the primary, isolation, or retention ground, or are explicitly connected to ground pins of leaf cells. In these cases, you can derive the ground type for these supply nets and later implement them as real ground nets by using the **derive_pg_connection -create_net** command.

By default, nothing is set for this variable, which means no supply nets are explicitly identified as ground nets.

The names listed in this variable must be actual supply net object names in the design. If any names do not match an existing supply net object name, those names are ignored, and you get a warning message. Furthermore, each listed name must be used only as a ground net,

not a power supply net, in the design. For example, if you use a listed net as a primary power supply, the **check_mv_design** or **derive_pg_connection** will report a **MV-544** conflict error, and no PG net will be created.

SEE ALSO

```
power_nets_for_upf_supply(3)
derive_pg_connection(2)
check_mv_design(2)
```

groute_VABoundaryToLSWeight

DESCRIPTION

The **groute_VABoundaryToLSWeight** variable is used in multivoltage mode. It specifies the value to adjust the cost for a level-shifter connection when switching voltage area from one to another.

SYNTAX

EXAMPLE

groute_avoidCouplingUser

DESCRIPTION

The **groute_avoidCouplingUser** variable enables the global router to space nets during routing. The nets to be considered need to be set with the Set Net Constraints command. If this variable is set to 1, timing spacing will be honored.

SYNTAX

EXAMPLE

groute_avoidXtalk

DESCRIPTION

The **groute_avoidXtalk** variable turns on/disables crosstalk prevention during global routing. If this variable is set to 1, the global router tries to avoid assigning nets with coupling to the same gcell.

SYNTAX

EXAMPLE

groute_blncdToSkewCntrlRatio

DESCRIPTION

The **groute_blncdToSkewCntrlRatio** variable sets a threshold for turning on the balanced skew control mode. For nets with an aspect ratio smaller than the variable value, balanced skew control mode is enabled. The aspect ratio is calculated based on the bounding box formed by enclosing the pins.

SYNTAX

EXAMPLE

groute_blockEdgeAccess

DESCRIPTION

The **groute_blockEdgeAccess** variable allows the global router to access pins on edges of hard macros. By default, the global router reaches pins on edges of hard macros. When the variable is set to 1, it allows the global router is allowed to access or drop vias on whole macro pins without edges limitation.

SYNTAX

EXAMPLE

groute_brokenNetsThresholdPercent

DESCRIPTION

The **groute_brokenNetsThresholdPercent** variable specifies a threshold for incremental global routing to be performed. This is effective when incremental is set to 1.

SYNTAX

EXAMPLE

groute_clockBalanced

DESCRIPTION

The **groute_clockBalanced** variable turns balanced routing on clock nets on and off.

SYNTAX

EXAMPLE

groute_clockComb

DESCRIPTION

The **groute_clockComb** variable enables the global router to connect ports directly to the prerouted clock trunks to minimize clock skew. Be aware that this requires greater routing resources, if the clock pins are not close to the clock trunks. The Comb mode works on designs with pre-existing clock nets.

SYNTAX

EXAMPLE

groute_combDistance

DESCRIPTION

The **groute_combDistance** variable sets a distance threshold, in gcell units, for clock routing to connect clock pins directly to clock nets.

SYNTAX

EXAMPLE

groute_combMaxConnections

DESCRIPTION

The **groute_combMaxConnections** variable limits the number of direct connections allowed to the clock pins from the same clock trunk. The default value of -1 allows a clock net to be connected to any number of clock pins.

SYNTAX

EXAMPLE

groute_compactMode

DESCRIPTION

The **groute_compactMode** variable determines the size of gcell used for global routing. If it is set to 0, gcells of 1 cellrow height will be created. If it is set to 1, the global router automatically adjusts the global route cell size.Increasing the size of gcell will complete the global routing faster at the cost of quality as routing becomes coarser.

SYNTAX

EXAMPLE

groute_congestionWeight

DESCRIPTION

The **groute_congestionWeight** variable specifies the relative importance of routing congestion versus wire length. As the variable value increases, the router tries harder to avoid routing congestion at the cost of increased wire length.

SYNTAX

EXAMPLE

groute_densityDriven

DESCRIPTION

The **groute_densityDriven** variable turns density-driven global routing on or off. In the default mode, the router decides the weight given to density relative to wire length during global routing. If timing driven or crosstalk is turned on, global route turns on the **groute_densityDriven** automatically.

SYNTAX

EXAMPLE

groute_detourLimitMinNetLen

DESCRIPTION

The **groute_detourLimitMinNetLen** variable controls the way the global router implements **maxDetourPercent** on nets. If the value is set to 0, the router forces **maxDetourPercent** on all nets. If the value is set to n, **maxDetourPercent** is applied only to nets of length greater than n gcells.

SYNTAX

EXAMPLE

groute_extraCostsApplyPercent

DESCRIPTION

The **groute_extraCostsApplyPercent** variable applies the **wireCost** and **viaCost** settings to the top n percent of nets in the design. The **wireCost** and **viaCost** variables are route variables. The route extra cost applies to global routing, track assignment and detail routing. This variable applies the extra cost percentage only for global route.

SYNTAX

EXAMPLE

groute_extraWireLengthOpt

DESCRIPTION

The **groute_extraWireLengthOpt** variable instructs the global router to run an additional rerouting phase to reduce wire length on nets that have no congestion. It does not perform any optimization.

SYNTAX

EXAMPLE

groute_forceUpperLayersForCritNets

DESCRIPTION

The **groute_forceUpperLayersForCritNets** variable specifies the mode for upper layer usage. To improve timing, the tool will route timing-critical nets on upper layers that have lower RC.

SYNTAX

EXAMPLE

groute_horReserveTracks

DESCRIPTION

The **groute_horReserveTracks** variable determines the number of horizontal free tracks reserved in each gcell.

SYNTAX

EXAMPLE

groute_ignoreViaBlockage

DESCRIPTION

The **groute_ignoreViaBlockage** variable specifies the mode to honor a via blockage. By default, global route ignores via blockage.

SYNTAX

EXAMPLE

groute_incremental

DESCRIPTION

The **groute_incremental** variable controls whether global router runs groute_incrementally.

SYNTAX

EXAMPLE

groute_macroBndryDir

DESCRIPTION

The **groute_macroBndryDir** variable controls the use of **macroBndryTrkUtil** for layers in different directions. If the value is set to 1, **macroBndryTrkUtil** will be applied to layers in both directions.

SYNTAX

EXAMPLE

groute_macroBndryExt

DESCRIPTION

The **groute_macroBndryExt** variable defines extension (in g-cells) of macro boundary used in **macroBndryTrkUtil**. The value is from -5 to 20, and default value is -1.

SYNTAX

EXAMPLE

groute_macroBndryTrkUtil

DESCRIPTION

The **groute_macroBndryTrkUtil** variable limits the utilization of tracks available in the gcells near a macro boundary to a specified percentage. This variable is used to control the accessibility of pins and congestion at the macro boundaries. By default, the router uses 100 percent of available tracks in the macro boundary width.

SYNTAX

EXAMPLE

groute_macroBndryWidth

DESCRIPTION

The **groute_macroBndryWidth** variable specifies a distance from the corners of macros. Within this distance, the global router obeys the limits on track utilization specified by **macroCornerTrkUtil** and **macroBndryTrkUtil**. By default, one row or column of gcells is considered from the corners of the macro.

SYNTAX

EXAMPLE

groute_macroCornerTrkUtil

DESCRIPTION

The **groute_macroCornerTrkUtil** variable limits the utilization of tracks available in the gcells near a macro corner to a specified percentage. This variable is used to control the accessibility of pins and congestion at the macro corners. By default, the router uses 100 percent of available tracks in the macro boundary width.

SYNTAX

EXAMPLE

groute_mapOnly

DESCRIPTION

The **groute_mapOnly** variable specifies whether the router generates the congestion map based on global routing without creating the global wires.

SYNTAX

EXAMPLE

groute_maxDetourPercent

DESCRIPTION

The **groute_maxDetourPercent** variable directs the global router to have no more than the specified percentage of detours on any net. If the value is set to -1, the router is free to make any number of detours (or none).

SYNTAX

EXAMPLE

groute_netCriticality

DESCRIPTION

The **groute_netCriticality** variable determines the order in which the global router routes the nets during initial route. Net criticality can be set on nets in the design by using the Scheme function **dbSetNetCriticality**. If this variable is set to 1, the global router first routes the nets with higher criticality value. If the value is set to 0, net criticality has no effect on routing order. However, net criticality always has an effect on the congestion cost.

SYNTAX

EXAMPLE

groute_noTopLevelBusFeedThroughs

DESCRIPTION

The **groute_noTopLevelBusFeedThroughs** variable determines whether feedthroughs are allowed on bus signals.

SYNTAX

EXAMPLE

groute_paEqPinNetMaxPort

DESCRIPTION

Setting the **groute_paEqPinNetMaxPort** variable to 1 creates equivalent pins for nets with no more than n ports.

SYNTAX

EXAMPLE

groute_powerDriven

DESCRIPTION

The **groute_powerDriven** variable turns power-driven global routing on or off.

SYNTAX

EXAMPLE

groute_rcOptByLength

DESCRIPTION

The **groute_rcOptByLength** variable controls the choice of layers for global router to reduce RC. The global router chooses layers with lower RC values based on the value of this variable. This variable is only active when the timing driven global route option is enabled.

SYNTAX

EXAMPLE

groute_reportDemandOnly

DESCRIPTION

The **groute_reportDemandOnly** variable specifies the mode to report demand only. In this mode, the tool performs virtual route only, with no rerouting phases. In the log file, it reports the average gcell capacity per layer.

SYNTAX

EXAMPLE

groute_reportEffectiveOverflow

DESCRIPTION

The **groute_reportEffectiveOverflow** variable specifies the mode for generating an effective overflow report.

SYNTAX

EXAMPLE

groute_reportGCellDensity

DESCRIPTION

The **groute_reportGCellDensity** variable controls reporting of gcell density for each layer. In density-driven mode, the router generates a report of gcell density. However, this variable is independent of the density-driven switch.

SYNTAX

EXAMPLE

groute_reportNetOrdering

DESCRIPTION

The **groute_reportNetOrdering** variable specifies the number of nets to be reported according to the routing order. By default, the global router does not report on net ordering. If this variable is set to n, global route will report the first n nets in the order of routing.

SYNTAX

EXAMPLE

groute_reserveTracksForPowerFile

DESCRIPTION

The file indicated by the **groute_reserveTracksForPowerFile** variable is used to define the percentage of routing tracks on each layer that are reserved for power routing. For example, Suppose the **reservedTracks.rc** file defines METAL1 layer as 20. Then, the router reserves 20 percent of available routing tracks in METAL1 for power routing later. Therefore, global route uses only 80 percent of the routing tracks.

SYNTAX

EXAMPLE

groute_skewControl

DESCRIPTION

The **groute_skewControl** variable turns skew control on or off during global routing. If the value is set to 1, the global router tries to minimize the gross delay in net skew. Skew control applies to all signal nets (except for small nets) but skew control for clock nets occurs only when **clockBalanced** is set to 1.

SYNTAX

EXAMPLE

groute_skewControlWeight

DESCRIPTION

The **groute_skewControlWeight** variable determines the importance given to skew control on the net during global routing. You can set the skew control weight from 1 to 10 based on net criticality.

SYNTAX

EXAMPLE

groute_speed

DESCRIPTION

The **groute_speed** variable specifies the effort at which the global router should run. The global router runs a different number of phases, depending on the specified value. It is recommended you run the router in default mode.

SYNTAX

EXAMPLE

groute_timingDriven

DESCRIPTION

The **groute_timingDriven** variable turns timing-driven global routing on or off. The **timingWeight** variable controls the trade-off between timing and wire length during global routing. By default, timing-driven mode is turned off.

SYNTAX

EXAMPLE

groute_timingWeight

DESCRIPTION

The **groute_timingWeight** variable sets the weight given to the timing relative to wire length during global routing. This variable is effective only in timing-driven mode.

SYNTAX

EXAMPLE

groute_turboMode

DESCRIPTION

The **groute_turboMode** variable is used to improve runtime.

SYNTAX

EXAMPLE

groute_verReserveTracks

DESCRIPTION

The **groute_verReserveTracks** variable determines the number of vertical free tracks reserved in each gcell.

SYNTAX

EXAMPLE

groute_xtalkWeight

DESCRIPTION

The **groute_xtalkWeight** variable defines the weight given to crosstalk prevention during global routing.

SYNTAX

EXAMPLE

gui_build_query_data_table

Controls whether the tool initializes data for fast data query after the GUI starts.

TYPE

Boolean

DEFAULT

The default value for **gui_build_query_data_table** is true.

DESCRIPTION

This variable controls whether IC Compiler initializes query data during GUI startup. By default, the **gui_build_query_data_table** variable is true and IC Compiler initializes query data, which might require some time for a large design. You can set the variable to false to prevent the tool from initializing query data.

EXAMPLES

The following example prevents the tool from initializing query data.

```
prompt> set gui_build_query_data_table false
```

SEE ALSO

`printvar(2)`

gui_custom_setup_files

Variable for specifying GUI customization files to be loaded during GUI startup.

TYPE

Tcl list of fully-qualified file names

DEFAULT

$synopsys/admin/setup/.synopsys_<app>_gui.tcl

DESCRIPTION

This variable specifies a set of files that should be sourced when the GUI starts up. You can add this variable setting to the application setup file to provide GUI customizations to individuals, or share customizations among a group of users. A GUI customization file typically contains commands to specify hotkeys, menus, or toolbars which implement specific functions to support a custom environment or flow.

The GUI initializes and searches the list of files in order. Each file in the list is sourced. Finally, your .synopsys_<app>_gui.tcl file is loaded to complete the customizations.

You can disable the loading of the customizations by setting the **gui_disable_custom_setup** variable.

SEE ALSO

```
printvar(2)
gui_disable_custom_setup(3)
```

gui_default_window_type

,IP **gui_default_window_type** Read-only variable specifying the default window type for GUI customization commands.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This read-only variable contains the name of the window type that is used as the default when a menu, hotkey, or toolbar customization is specified without specifying a window type explicitly.

SEE ALSO

```
gui_create_menu(2)
gui_create_toolbar(2)
gui_create_toolbar_item(2)
gui_delete_menu(2)
gui_delete_toolbar(2)
gui_delete_toolbar_item(2)
gui_report_hotkeys(2)
gui_set_hotkey(2)
printvar(2)
```

gui_disable_abstract_clock_graph

Specifies whether the levelized and latency clock graphs use abstraction.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable specifies whether the levelized and latency clock graphs use the abstraction functionality. By default (**false**), the levelized and latency clock graphs use abstraction. When you set this variable to **true**, the levelized and latency clock graphs do not use abstraction. You must set this variable before invoking the GUI; changes to this variable have no effect after the GUI is started.

SEE ALSO

`start_gui(2)`

gui_disable_custom_setup

Variable for disabling gui customizations

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable specifies whether GUI customization loading is disabled during GUI startup. Set the variable to true to prevent GUI customizations from loading. This includes customizations specified in the **gui_custom_setup_files** variable as well as your .synopsys_<app>_gui.tcl file.

SEE ALSO

```
printvar(2)
gui_custom_setup_files(3)
```

gui_online_browser

Specifies the name of the browser used to invoke the online help system from the help menu of the product.

TYPE

string

DEFAULT

netscape

GROUP

gui

DESCRIPTION

This string variable holds the value of the default browser which is used to invoke online help system from Help menu of the application. The value the variable should be either **netscape**, **mozilla** or **firefox**.

If you specify a browser other than those listed above, the application defaults to the netscape browser.

Use the following command to determine the current value of the variable:

```
prompt> printvar gui_online_browser
```

SEE ALSO

`gui_custom_setup_files(3)`

hdlin_auto_save_templates

Controls whether HDL designs containing parameters are read in as templates.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

When set to **true** this variable reads in HDL designs containing parameters as templates. HDL parameter files include Verilog modules with parameter declarations and VHDL entities with generic declarations.

When an HDL file is read in as a template, it can be manipulated with the **printvar -templates**, **remove_template**, and **elaborate** commands. The template can also be built automatically when instantiated from another HDL design.

When the value is **false** (the default), HDL designs are read in and built instantly. This can be overridden for a design with a template pseudo comment.

For Verilog, place the following comment anywhere in the module:

```
// synopsys template
```

For VHDL, place the following comment in the entity after the **port** declaration:

```
-- synopsys template
```

To determine the current value of this variable, use the **printvar hdlin_auto_save_templates** command. For a list of HDL variables and their current values, use the **print_variable_group hdl** command.

SEE ALSO

```
template_parameter_style(3)
template_separator_style(3)
```

hdlin_autoread_exclude_extensions

Specifies the file extensions of files that you do not want the **analyze** command or **read_file** command to analyze when using the **-autoread** option.

TYPE

Tcl list of strings

DEFAULT

""

GROUP

hdl_variables

DESCRIPTION

This variable specifies the file extensions of files that you do not want the **analyze** and **read_file** commands to analyze when you specify the **-autoread** option.

Use **hdlin_autoread_exclude_extensions** to exclude files with certain extensions.

This Tcl variable supersedes the values of the **hdlin_autoread_verilog_extensions**, **hdlin_autoread_sverilog_extensions** and **hdlin_autoread_vhdl_extensions** language extension variables.

The following examples show how the variable is used:

```
prompt> set hdlin_autoread_exclude_extensions { .inc .v_in .def }
prompt> set hdlin_autoread_exclude_extensions { .h .H .hh .c .C .cc }
prompt> set hdlin_autoread_exclude_extensions { .txt .doc }
```

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

SEE ALSO

```
read_file(2)
hdlin_autoread_verilog_extensions(3)
hdlin_autoread_sverilog_extensions(3)
hdlin_autoread_vhdl_extensions(3)
```

hdlin_autoread_sverilog_extensions

Specifies the file extensions of files analyzed as SystemVerilog source code by the **analyze** and **read_file** commands with the **-autoread** option.

TYPE

Tcl list of strings

DEFAULT

".sv .sverilog"

GROUP

hdl_variables

DESCRIPTION

This variable specifies the file extensions of files analyzed as SystemVerilog source code by the **analyze** and **read_file** commands with the **-autoread** option. Set the value of this Tcl variable to a list of file extensions to specify which files will be inferred as SystemVerilog source files. When the **analyze** command or **read_file** command collect SystemVerilog source files from the specified source code location, all files that end with one of the strings in this list are analyzed as SystemVerilog files.

If any of the **hdlin_autoread_sverilog_extensions** values match any of the **hdlin_autoread_verilog_extensions** or **hdlin_autoread_vhdl_extensions** variable values, an AUTOREAD-205 error is issued.

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

SEE ALSO

```
hdlin_autoread_exclude_extensions(3)
hdlin_autoread_verilog_extensions(3)
hdlin_autoread_vhdl_extensions(3)
```

hdlin_autoread_verilog_extensions

Specifies the file extensions of files analyzed as Verilog source code by the **analyze** and **read_file** commands with the **-autoread** option.

TYPE

Tcl list of strings

DEFAULT

".v"

GROUP

hdl_variables

DESCRIPTION

This variable specifies the file extensions of files analyzed as Verilog source code by the **analyze** and **read_file** commands with the **-autoread** option. Set the value of this Tcl variable to a list of file extensions to specify which files will be inferred as Verilog source files. When the **analyze** command or **read_file** command collect Verilog source files from the specified source code location, all files that end with one of the strings in this list are analyzed as Verilog files.

If any of the **hdlin_autoread_verilog_extensions** values match any of the **hdlin_autoread_sverilog_extensions** or **hdlin_autoread_vhdl_extensions** variable values, an AUTOREAD-205 error is issued.

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

SEE ALSO

```
hdlin_autoread_exclude_extensions(3)
hdlin_autoread_sverilog_extensions(3)
hdlin_autoread_vhdl_extensions(3)
```

hdlin_autoread_vhdl_extensions

Specifies the file extensions of files analyzed as VHDL source code by the **analyze** and **read_file** commands with the **-autoread** option.

TYPE

Tcl list of strings

DEFAULT

".vhd .vhdl"

GROUP

hdl_variables

DESCRIPTION

This variable specifies the file extensions of files analyzed as VHDL source code by the **analyze** and **read_file** commands with the **-autoread** option. Set the value of this Tcl variable to a list of file extensions to specify which files will be inferred as VHDL source files. When the **analyze** command or **read_file** command collect VHDL source files from the specified source code location, all files that end with one of the strings in this list are analyzed as VHDL files.

If the variable is not defined, or if it is set to an empty list, only files with the following extensions are inferred as VHDL source files: .vhd .vhdl

If any of the **hdlin_autoread_vhdl_extensions** values match any of the **hdlin_autoread_verilog_extensions** or **hdlin_autoread_sverilog_extensions** variable values, an AUTOREAD-205 error is issued.

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

SEE ALSO

```
hdlin_autoread_exclude_extensions(3)
hdlin_autoread_verilog_extensions(3)
```

```
hdlin_autoread_sverilog_extensions(3)
```

hdlin_enable_assertions

Controls the tool's use of SystemVerilog Assertions.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Setting **hdlin_enable_assertions** to **true** allows the **elaborate** and **read_sverilog** commands to process a synthesis-friendly subset of SystemVerilog RTL Assertions. Facts added via assertions can contribute to optimization by ruling out machine states that cannot arise during normal operation of the design. This switch controls actions taken only during the elaborate phase, including elaborations that occur during linkage or compilation.

To be activated, an assertion statement must belong to a narrow category of synthesizable deferred immediate assertions. It must have a dedicated statement label that gives it a unique, lexically scoped name. It also must be analyzed in **sverilog** format (or by **read_sverilog**).

The asserted expression should compare only one variable or signal to a compile-time constant, or else it should make a $onehot or $onehot0 claim about a collection of signal wires.

Consult the *HDL Compiler for SystemVerilog User Guide* for the exact definition of the subset currently supported by this release. All labeled assertions within this subset survive analysis,

To confirm that an assertion should be considered during optimization do the following:

- Set **hdlin_enable_assertions true** when it is being elaborated.

- Set the lexically scoped name of its statement label as an element of the **confirmed_SVA()** array variable in the tool environment in which the assertion is elaborated. Set it to the value of **true**.

When the above conditions are met, the labeled and confirmed assertions encoded in the intermediate (or template) design files can begin to influence logic minimization done by the tool.

Note that either linking or elaborating designs with unresolved template references can cause subdesign elaborations whose confirmed assertions may also be exploited by the tool's optimizations. The following is an example that confirms 3 assertions named *assertion_label1*, *global_assertion1*, and *assumption_about_port*:

```
analyze -f sverilog my_file.sv
# The following lines can be brought in via a source command
set confirmed_SVA(module1.scope1.assertion_label1) true
set confirmed_SVA(module1.global_assertion1) true
set confirmed_SVA(module2.generated_scope[0].assumption_about_port) true

set hdlin_enable_assertions true
elaborate module2
```

SEE ALSO

```
ELAB-333(n)
ELAB-414(n)
```

hdlin_enable_elaborate_ref_linking

Controls the hierarchical elaboration flow support by SystemVerilog.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

This variable, when set to **true**, enables hierarchical elaboration support by HDL Compiler SystemVerilog. It allows the **elaborate** command to include additional contextual linkage information embedded into the elaborate designs, such as typeparams, parameter, and interface specializations.

Later, during final top module elaboration, the switch allows the linker to reach and analyze this additional contextual linkage information stored in designs, helping the pre-elaborated designs to be matched with the top module instantiations, enabling the hierarchical elaboration in a bottom-up flow.

For more information, see the *HDL Compiler for SystemVerilog User Guide*.

SEE ALSO

hdlin_enable_hier_naming

Controls if the registers and instances in a design are named with their hierarchical path attached or not.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

This variable controls if registers and instances are named in the hierarchical way or not. For example,

```
      4 generate for (i=0; i < 2; i = i + 1 ) begin : mygenblk
5   always @(posedge clk or negedge rst) begin : myblk      6
reg myreg;      7     if( ! rst ) begin      8       qout = 0;
9       myreg = 0;     10     end     11     else begin     12
qout = myreg;     13       myreg = datain;     14     end     15
end     16 end endgenerate
```

If **hdlin_enable_hier_naming** is set to *true*, the registers inferred from the code on line 6 will be 'mygenblk[0].myblk.myreg_reg' and 'mygenblk[1].myblk.myreg_reg'.

To determine the current value of this variable, use the **printvar hdlin_enable_hier_naming** command. For a list of HDL variables and their current values, use **print_variable_group hdl**.

SEE ALSO

hdlin_enable_relative_placement

Enables RTL relative placement support.

TYPE

string

DEFAULT

rb

GROUP

hdl_variables

DESCRIPTION

This variable determines whether and how the tool supports RTL relative placement. Valid values for **hdlin_enable_relative_placement** are **mux**, **rb**, **rb;mux**, **mux;rb**, and **none**.

- When the value is **mux**, relative placement is enabled for MUX_OPs.
- When the value is **rb**, relative placement is enabled for register banks.
- When the value is **rb;mux** or **mux;rb**, relative placement is enabled for register banks and MUX_OPs.
- When the value is *none*, the tool disables relative placement for all of the structures.

Relative placement for MUX_OPs is not supported if it violates the **hdlin_mux_rp_limit** variable.

To determine the current value of this variable, use the **printvar hdlin_enable_relative_placement** command. For a list of HDL variables and their current values, use **print_variable_group hdl**.

SEE ALSO

`hdlin_mux_rp_limit(3)`

hdlin_interface_port_ABI

Chooses a linkage convention for SystemVerilog ports of type interface or modport.

TYPE

integer

DEFAULT

3

GROUP

hdl_variables

DESCRIPTION

The **hdlin_interface_port_ABI** variable is effective only when HDL Compiler (Presto) parses the source code of an **interface** or **module** declaration in SystemVerilog format (**-f sverilog**) during an **analyze** or **read** command. Its setting selects the linkage conventions to be used when interface instances or ports connect across a subdesign port. Because this setting is recorded when the interface is analyzed, its author can determine interface-specific aspects the ABI conventions once for all users, independent of any other interfaces they might also use. Since the setting is also recorded by modules that form and pass along arrays of interface instances, it allows a module which creates arrays to specify the conventions for connecting them to its subdesigns.

The memory elements and signal wires synthesized specifically for each interface instance connect to physical input, output, or inout ports at every hierarchical boundary where that interface instance connects to other design elements through an interface port or modport. The "Application Binary Interface" (ABI) is a set of conventions for which connections to make, their positional order and port names, and how to treat the contents of arrayed interface ports. Both designs being connected must agree on the ABI of their actual and formal interface ports. This agreement occurs automatically when both interface clients are elaborated using identical versions of the interface, analyzed with the same choice of ABI setting. Usually it suffices to analyze the interface once into a design library (such as "WORK") shared by all its clients as they are elaborated.

The integer value of **hdlin_interface_port_ABI** encodes a bit vector with one bit position for each ABI feature. Its default value is 3'b011.

bit[0] Canonical Bounds (ABI 3'b??0 vs 3'b??1)

Bit 0 only affects arrayed ports (of subdesign reference cells) whose elements are instances (or modports) of some interface definition. The ABI bit[0] of the current module controls, not the ABI of the interface or the down design. The effect is noticeable only when the interface array actual port expression has bounds that differ from the canonical bounds for unpacked arrays (such as [*n*] or [0:*n*-1]) and the actual port is passed by .*portname*(...). In these cases, the array index values are explicit in the port names being matched by Design Compiler's linker.

- An interface array connected from a module whose ABI value is **0** in bit position 0 uses the actual array expression's bounds to invent port names of the reference cell. Note that different instantiations of the same down design might then require distinct formal port names, and these will be produced by multiple elaborations of the down design.
- An interface array connected from a module whose ABI value is **1** in bit position 0 (this is the default) uses canonical bounds to invent port names of the reference cell. Note that part-selects of unpacked arrays always produce canonical bounds, so this option avoids the primary source of multiple elaboration due to bounds-matching.
- Note that with either setting, formal port names can differ from the internal net names to which they connect. In fully linked designs, this produces net names in the down design that are taken from their connectivity with the top design. The canonical setting exactly resembles the names produced during linkage for all other forms of bused ports

bit[1] Demoted Modports (ABI 3'b?0? vs 3'b?1?)

Bit 1 affects only those port connections where the formal interface port specifies a modport and the actual interface instance does not.

When the value in bit position 1 is **1** (the default), these connections are successfully linked per the IEEE-1800 standard definition. This is accomplished by adding dummy (unused) formal ports to a special elaboration of the design that correspond in name and position to all interface content that lies outside the given formal modport. This elaboration uses a different **-param** string than that produced for a design reference with matching modport specifications for this interface.

When the value in bit position 1 is **0**, no special elaboration is created to accommodate formal-only modport selection; no dummy formal ports are added. In general, this will prevent linkage of an actual interface instance to a formal modport selected by the down design.

Note that the default (demote) setting allows interface modport connections to be made using .* and .portname forms as the actual port. The cost of this convenience is the introduction of unused design ports.

bit[2] API-style (ABI 2'b0?? vs 3'b1??)

Bit positions 2 and above are reserved for future extensions of the interface port ABI. Premature use of these settings is not supported.

SEE ALSO

hdlin_mux_for_array_read_sparseness_limit

Prevents inference of a sparse multiplexer for array read operations when the percentage of MUX_OP data inputs that are connected is below **hdlin_mux_for_array_read_sparseness_limit**.

TYPE

integer

DEFAULT

90

GROUP

hdl_variables

DESCRIPTION

This variable prevents inference of a sparse multiplexer for array read operations when the percentage of MUX_OP data inputs that are connected is below **hdlin_mux_for_array_read_sparseness_limit**.

For array read operations, HDL Compiler can infer a MUX_OP or a SELECT_OP. MUX_OPs can be mapped to MUX cells in the technology library but SEL_OPs cannot be mapped in this manner. MUX cells might be desirable for lowering congestion or implementing specific switching behavior. For an array read operation, if the array size is not a power of 2, HDL Compiler uses a MUX_OP whose size is the next power of 2. The unused data inputs are disconnected. This is called a sparse multiplexer.

The **hdlin_mux_for_array_read_sparseness_limit** variable guides the HDL Compiler to choose the right cell for an array read operation. This value corresponds to the lowest percentage of MUX_OP data inputs that must be connected. If the percentage is below this limit, HDL Compiler infers a SELECT_OP instead, whose number of data inputs does not need to be a power of 2, and therefore gives lower area.

The default percentage is 90. This means that if at least 90% of MUX_OP data inputs are connected, a MUX_OP is inferred. If this percentage is below 90%, a SELECT_OP is inferred instead.

The following are examples of code that will not infer a MUX_OP:

```
VHDL
----
signal a : bit_vector (0 to 99);
signal x: bit;
  ...
x <= a(i);

Verilog
-------
reg [0:99] a;
reg x;
  ...
x = a[i];
```

In both examples, a MUX_OP of 128 data inputs is required, but only 100 would be connected. This gives a percentage of 100/128*100; that is, only 78% of MUX_OP data inputs are connected. As this is below 90%, a SELECT_OP will be inferred.

Note that this variable prevents sparse MUX_OP inference for array read operations only. Inference of MUX_OP for if, case, and ?: operations is governed by the **hdlin_infer_mux**, **hdlin_mux_size_limit** and **hdlin_mux_oversize_ratio** variables.

SEE ALSO

hdlin_mux_rp_limit

Limits the minimum bit width and minimum number of inputs for a relative placement multiplexer.

TYPE

string

DEFAULT

128x4

GROUP

hdl_variables

DESCRIPTION

This variable limits the minimum bit width and minimum number of inputs for a relative placement multiplexer.

The value is *mxn* where *m* is the minimum bit width and *n* is the minimum number of inputs for the multiplexer. The default value is 128x4, meaning that HDL Compiler does not generate relative placement MUX_OPs for multiplexers specified with fewer than 4 inputs and a bit width of less than 128.

The time it takes to process a relative placement MUX is proportional to the value of **hdlin_mux_rp_limit** squared. For example, the time it takes to process a 128x64 MUX for relative placement is approximately four times that for processing a 128x32 MUX.

To determine the current value of this variable, use the **printvar hdlin_mux_rp_limit** command. For a list of HDL variables and their current values, use **print_variable_group hdl**.

SEE ALSO

`hdlin_enable_relative_placement(3)`

hdlin_mux_size_only

Controls which MUX_OP cells receive the **size_only** attribute in HDL Compiler.

TYPE

integer

DEFAULT

1

GROUP

hdl_variables

DESCRIPTION

To ensure that MUX_OP GTECH cells are mapped to MUX technology cells, you must apply a **size_only** attribute to the cells to prevent logic decomposition in later optimization steps. Beginning with the B-2008.09-SP3 release, you can control which MUX_OP cells receive the **size_only** attribute by using the **hdlin_mux_size_only** variable. The following options are valid for **hdlin_mux_size_only**:

0

Specifies that no cells receive the **size_only** attribute.

1

Specifies that MUX_OP cells that are generated with the RTL **infer_mux** pragma and that are on set and reset signals receive the **size_only** attribute.

2

Specifies that all MUX_OP cells that are generated with the RTL **infer_mux** compiler directive receive the **size_only** attribute.

3

Specifies that all MUX_OP cells on set and reset signals receive the **size_only** attribute; for example, MUX_OP cells that are generated by the **hdlin_infer_mux** variable set to **all**.

4

Specifies that all MUX_OP cells receive the **size_only** attribute; for example, MUX_OP cells that are generated by the **hdlin_infer_mux** variable set to **all**.

To determine the current value of this variable, use the **printvar hdlin_mux_size_only** command. For a list of HDL variables and their current values, use **printvar_variable_group hdl**.

Note that all MUX_OP cells inferred using the **infer_mux_override** directive will receive the **size_only** attribute.

SEE ALSO

`set_size_only(2)`

hdlin_reporting_level

Determines which information Presto HDL Compiler prints in the report.

TYPE

string

DEFAULT

basic

GROUP

hdl_variables

DESCRIPTION

This variable controls the amount of output information to be included in the Presto elaboration report.

The following information is under control:

- **floating_net_to_ground** prints the report for floating net connects to ground. This variable is best used in conjunction with the **set hdlin_keep_signal_name user** command and is not guaranteed to report all nets. Use the **check_design** command for detecting unconnected pins and ports.
- **fsm** prints the report for inferred state variables.
- **inferred_modules** prints the report for inferred sequential elements.
- **mux_op** prints the report for MUX_OPs.
- **syn_cell** prints the report for synthetic cells.
- **tri_state** prints the report for inferred tristate elements.

The **hdlin_reporting_level** variable can be set to 4 base settings: **none**, **basic**, **comprehensive**, and **verbose**, as shown in the following table:

```
Table 1 Base Settings
```

Information included in report	none	basic	comprehensive	verbose
floating_net_to_ground	false	false	true	true
fsm	false	false	true	true
inferred_modules	false	true	true	verbose
mux_op	false	true	true	true
syn_cell	false	false	true	true
tri_state	false	true	true	true

In addition to the base settings above, you can also modify the base settings to have fine grain control of individual reports by either adding (+) or subtracting (-) specific report(s) from the base setting with the following keywords:

```
floating_net_to_ground
fsm
syn_cell
mux_op
inferred_modules
tri_state
```

EXAMPLES

The following example uses **comprehensive-fsm**:

```
set hdlin_reporting_level comprehensive-fsm
```

The generated report shows the following settings:

```
floating_net_to_ground  true
fsm                     false
inferred_modules        true
mux_op                  true
syn_cell                true
tri_state               true
```

The following example uses **verbose-mux_op-tri_state**:

```
set hdlin_reporting_level verbose-mux_op-tri_state
```

The generated report shows the following settings:

```
floating_net_to_ground  true
fsm                     true
inferred_modules        verbose
```

```
mux_op                  false
syn_cell                true
tri_state               false
```

The following example shows two commands that generate equivalent reports:

```
set hdlin_reporting_level basic+floating_net_to_ground+syn_cell+fsm

set hdlin_reporting_level comprehensive
```

To determine the current value of this variable, use the **printvar hdlin_reporting_level** command. For a list of HDL variables and their current values, use **printvar_variable_group hdl**.

SEE ALSO

hdlin_strict_verilog_reader

Controls whether the Verilog Netlist Reader enforces strict IEEE-1364 language specification compliance.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

This variable controls whether the Verilog Netlist Reader errors or not on illegal verilog, like repeated identifiers.

When the value of this variable is **true**, constructs usually accepted by the verilog reader, such as instantiations with the same name as a wire, will trigger an error.

When the value is **false** (the default), no error is issued.

SEE ALSO

hdlin_sv_packages

Specifies whether and how System Verilog packages should be analyzed.

TYPE

string

DEFAULT

enable

GROUP

hdl_variables

DESCRIPTION

Specifies which, if any, semantics the **analyze** or **read_file** command should apply when a **-format sverilog** source file declares a package. The setting affects the analyze step of SystemVerilog package declarations or references; it has no effect during **elaborate**. All synthesizable package semantics (including VHDL packages) are supported by **elaborate**, **link**, and **compile**. The allowed values for **hdlin_sv_packages** are **none**, **chain**, **dont_chain** and the **enable** default setting.

The setting **none** prevents parsing of package declarations or references.

All other settings accept packages (declarations, references, and imports) as specified in section 19.2 of the IEEE-1800-2005 System Verilog standard. The default setting for SV package users is **enable**. This setting provides a synthesizable subset of packages that is compatible with the current release of Synopsys' VCS and Formality products.

Although the SV package standard was approved after the first commercial implementations had been released, there is only one known dialect incompatibility. If you receive a VER-934 informational message, you may need to choose a specific package definition to be compatible with your local simulation tool. The only difference is in how an **import** statement treats names imported into the topmost (global) scope of a package_declaration:

- The **dont_chain** setting prevents imported names from being re-exported to clients of the package being declared.

- The *chain* setting always re-exports names that are imported into the global scope of a package; they may all be imported by the intermediate package's clients. An imported name and its definition which are re-exported (or chained) will not collide or interfere with copies of themselves in those cases where several intermediate packages redistribute content they acquired from a common source package (provided they all acquire it from a compatible analyzed version of the same source file).

In both variants, an imported name can be used freely within a package declaration to implement the package's exported content. The analyzed result, a file named *package_identifier*.pvk, always contains a full copy of all imported content. A package's .pvk file can stand alone; it does not require its clients to access the .pvk files that supplied its imported ingredients (unlike source-level file inclusion).

The chaining issue only concerns whether imported names become explicitly visible to an intermediate package's clients as do objects explicitly declared at the outermost level of the intermediate package. Because a wildcard "import intermediate_pkg::*;" encumbers all of the exportable names found in "intermediate_pkg", the selection of **chain** or **dont_chain** can alter the outcome of name resolutions when several packages are combined in a downstream client.

The setting of **hdlin_sv_packages** at the time a package is analyzed is compiled into a visibility property on the imported, global names in the resulting .pvk file. This is an independent property that can be different at each level of a supply chain; it is not an inherited property of the name itself. A VER-934 informational message always indicates how this property is being set for those names where it may eventually matter.

For details, see the *IEEE-1800-2005 System Verilog Reference Manual*.

To determine the current value of this variable, use the **printvar hdlin_sv_packages** command. For a list of HDL variables and their current values, use **print_variable_group hdl**.

SEE ALSO

```
read_file(2)
VER-20(n)
VER-21(n)
VER-934(n)
```

hdlin_sverilog_std

Controls whether HDLC SystemVerilog enforces SystemVerilog 2005 or SystemVerilog 2009 or SystemVerilog 2012.

TYPE

integer

DEFAULT

2012

GROUP

hdl_variables

DESCRIPTION

This variable controls whether HDLC SystemVerilog is to enforce SystemVerilog 2005 or SystemVerilog 2009 or SystemVerilog 2012.

- If the variable is set to 2012, SystemVerilog 2012 is enforced. There is no Verilog 2012.
- If the variable is set to 2009, SystemVerilog 2009 is enforced. There is no Verilog 2009.
- If the variable is set to 2005 (the default), SystemVerilog 2005 is enforced.

The SystemVerilog 2009 standard merged two previous standards, Verilog 2005 and SystemVerilog 2005, which defined extensions to it. Those two standards were designed to be used as one language, so merging the base Verilog language and the SystemVerilog extensions into a single standard provides all information regarding syntax and semantics in a single document. Additionally, there are many extensions beyond SystemVerilog 2005 in SystemVerilog 2009.

The SystemVerilog 2012 standard adds extensions beyond SystemVerilog 2009.

SEE ALSO

hdlin_verification_priority

Instructs the tool to prioritize formal verification over QoR while reading the RTL files.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

Setting this variable to **true** adjusts optimization inside of the tool to prioritize formal verification compatibility over QoR. This variable only affects optimizations done during the reading and elaboration of RTL files. To control optimization during later stages, see the man page for the **set_verification_priority** command.

Set this variable in response to a hard verification from your formal verification tool.

SEE ALSO

hdlin_vhdl_syntax_extensions

Enables VHDL language features that are currently outside of the supported synthesizable subset.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

This variable enables VHDL language features that are currently outside of the supported synthesizable subset. Use of these language constructs must be accompanied by thorough verification.

The following features are currently enabled:

- Deferred constant definition: a constant declaration not accompanied by an assignment expression.
- Arrays of base type Boolean: use Boolean as the range type of an array.
- Impure functions: a function specification contains the reserved word impure.

SEE ALSO

hercules_home_dir

Specifies the path to the Hercules installation used for VUE to IC Compiler integration.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable specifies the version of Hercules that is used to load the Hercules extensions to the IC Compiler GUI. This extension provides additional functions for launching Hercules VUE and connecting it to the IC Compiler GUI.

If this variable is set (and it points to a valid Hercules installation), the Hercules extensions to the IC Compiler GUI are automatically loaded when the IC Compiler GUI is first invoked.

If you set the HERCULES_HOME_DIR environment variable, but you do not want the Hercules GUI extensions loaded into IC Compiler, set this variable to "" in your .synopsys_dc.setup file.

hier_dont_trace_ungroup

Disables ungroup tracing set on the design with the **ungroup** command.

TYPE

Boolean

DEFAULT

0

DESCRIPTION

This variable disables ungroup tracing set on the design with the **ungroup** command. When **hier_dont_trace_ungroup** is set to **0** (the default), the **ungroup** command places on the design being ungrouped a string attribute that describes the ungroup operation. Other tools (for example, RTL Analyzer) can later use the attribute to recreate the ungroup operation and trace between the mapped and GTECH (generic) circuits.

Setting **hier_dont_trace_ungroup** to **1** disables ungroup tracing and can increase the efficiency of **ungroup** and other commands.

SEE ALSO

`ungroup(2)`

high_fanout_net_pin_capacitance

Specifies the pin capacitance used to compute the loading of high-fanout nets.

TYPE

float

DEFAULT

1

GROUP

timing_variables

DESCRIPTION

This variable specifies the pin capacitance used to compute the loading of high-fanout nets.

The pin capacitance for high-fanout nets is computed by multiplying the capacitance specified by the **high_fanout_net_pin_capacitance** variable times the high-fanout threshold.

For best results the pin capacitance chosen should be large enough to cause violations on all constrained high-fanout nets. This forces the nets to be replaced with buffer trees during compilation.

To determine the current value of this variable, use the **printvar high_fanout_net_pin_capacitance** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

SEE ALSO

`high_fanout_net_threshold(3)`

high_fanout_net_threshold

Specifies the minimum number of loads for a net to be classified as a high-fanout net.

TYPE

integer

DEFAULT

1000

GROUP

timing_variables

DESCRIPTION

This variable specifies the minimum number of loads for a net to be classified as a high-fanout net.

Delays and loads of high-fanout are computed using a simplified model assuming a fixed fanout number. The rationale behind this is that delays of high-fanout nets are expensive to compute but such nets are often unconstrained (as in the case of global reset nets, scan enable nets, and so on). Those high-fanout nets that actually are constrained should eventually be replaced by buffer trees. So detailed delay calculations on such nets are expensive and usually unnecessary.

Setting the threshold to 0 (or to a very large number) ensures that no nets will be treated as high-fanout nets. However, be aware that forcing fully accurate delay calculations on high-fanout can significantly increase compilation runtime in some cases.

The pin capacitance for high-fanout nets is computed by multiplying the capacitance specified by the **high_fanout_net_pin_capacitance** variable times the high-fanout threshold plus the number of net drivers. If there are delay annotations remaining from an earlier flow step, these annotations will be used instead of the high-fanout net model.

The simplified net delay model is used only when computing data delays. Propagated clock latencies are always computed using the full accuracy net delay model.

To determine the current value of this variable, use the **printvar high_fanout_net_threshold** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

SEE ALSO

`high_fanout_net_pin_capacitance(3)`

icc_magnetpl_stop_after_seq_cell

Controls if sequential cells need to be pulled towards a specified magnet during **magnet_placement** the with **-stop_by_sequential_cells** option.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **icc_magnetpl_stop_after_seq_cell** variable controls whether sequential cells need to be pulled towards a specified magnet when using the **magnet_placement** command with the **-stop_by_sequential_cells** option. By default, the magnet placement operation is terminated before the sequential cell element when specified with the **-stop_by_sequential_cells** option.

SEE ALSO

```
get_magnet_cells(2)
magnet_placement(2)
```

icc_preroute_power_aware_optimization

Controls whether the preroute power-aware optimization flow is enabled.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether the preroute power-aware optimization flow is enabled.

By default (false), power is not considered during timing optimization when you run the **place_opt**, **clock_opt**, or **psynopt** command without the **-power** option.

If you set this variable to **true**, these commands perform power-aware timing optimization, even when you do not use the **-power** option.

Note that when you use the **-power** option with these commands, they perform power-aware timing optimization and leakage-power optimization, regardless of the setting of this variable.

This variable affects the following preroute optimization commands: **place_opt**, **clock_opt**, and **psynopt**.

SEE ALSO

```
place_opt(2)
clock_opt(2)
psynopt(2)
```

icc_preroute_tradeoff_timing_for_power_area

Controls whether timing, power and area tradeoff optimization is enabled for the **place_opt** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether timing, power and area tradeoff optimization is enabled for the **place_opt** command. The variable applies when you use the **place_opt** command with the **-power** and **-effort medium|high** options.

When you use the **place_opt -effort medium|high -power** command, by default, timing is treated as a more important QoR metric than power or area.

If you set this variable to **true**, the tool enables the tradeoff optimization which allows small timing degradations to improve power and area.

SEE ALSO

`place_opt(2)`

icc_snapshot_storage_location

Specifies the location for the create_qor_snapshot, remove_qor_snapshot, and report_qor_snapshot utilities.

TYPE

string

DEFAULT

snapshot

DESCRIPTION

This variable specifies the location in which to store snapshot data for the **create_qor_snapshot** command. The **report_qor_snapshot** command generates and reports snapshot data from this location. The **remove_qor_snapshot** command deletes snapshot(s) from this location.

SEE ALSO

```
create_qor_snapshot(2)
remove_qor_snapshot(2)
report_qor_snapshot(2)
```

ignore_binding_open_pins

Specifies the unconnected pins to be ignored during binding.

TYPE

string

DEFAULT

""

DESCRIPTION

Specifies the unconnected input pins to be ignored during binding. If you do not specify any pins, no unconnected pins are ignored.

You can specify a space-separated list of pin names or use an asterisk (*) to ignore all open pins during binding. You need to specify full hierarchical port names or leaf pin names for this variable.

You need to specify all the open pins that need to be ignored for binding before reading in the design netlist itself. After the design database is updated with the bound pins, this variable does not have any effect.

Note: This variable setting is not stored in the design database. You must set this variable in every session in your flow until you want binding of open pins to happen.

EXAMPLES

The following example shows that all unconnected input pins are ignored during binding.

```
prompt> set ignore_binding_open_pins "*"

Information: Unbinding all unconnected pin(s) in the design,
which was set to ignore binding by the 'ignore_binding_open_pins'
variable. (OPT-1155)
```

ignore_clock_input_delay_for_skew

Controls how clock skew calculations handle the input delay on clocks.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

This variable controls how clock skew calculations handle the input delay on clocks.

Normally, you should use the **set_input_delay** command only on data ports, and use the **set_clock_latency** command on clock ports. In cases where the **set_input_delay** command has been used on a clock port, the default behavior differs between PrimeTime and the synthesis timing engine used by Design Compiler and IC Compiler.

By default (**false**), the synthesis timing engine uses the input delay values set on the clocks with the **set_input_delay** command when computing the skew values.

When set to **true**, the synthesis timing engine ignores the input delay values set on the clocks with the **set_input_delay** command when computing the skew values of clocked registers, which is compatible with the default PrimeTime behavior.

Set this variable to **true** when correlating synthesis and Primetime results.

SEE ALSO

```
set_input_delay(2)
set_clock_latency(2)
```

ignore_guardband

Specifies if the power switches inserted honor or ignore the guardbands of other voltage areas while executing the **create_power_switch_array** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to *true*, the power switches inserted by the **create_power_switch_array** command ignore the guardbands of other voltage areas.

SEE ALSO

`create_power_switch_array(2)`

ignore_tf_error

Sets the flag for the tool to ignore unrecognized attributes in the technology file.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Setting the **ignore_tf_error** variable to **true** allows the tool to ignore unrecognized attributes when reading the technology file. Unrecognized attributes will still be reported using the TFCHK-009 error message. However, a TFCHK-096 informational message will be reported at the end of the technology file reading to indicate that these errors were ignored.

Since the technology file contains many attributes that are required for correct functionality of the tool, set this variable only after it has been verified that all of the unrecognized attributes in the technology file are safe to ignore.

SEE ALSO

`create_mw_lib(2)`

in_gui_session

This read-only variable is "true" when the GUI is active and "false" when the GUI is not active.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable can be used in writing Tcl code that depends on the presence the graphical user interface (GUI). The read-only variable has the value "true" if **gui_start** has been invoked and the GUI is active. Otherwise, the variable has the value "false" (default).

SEE ALSO

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

initial_target_library

Specifies the list of technology libraries of components to be used for the first part of leakage power optimization in **place_opt**.

TYPE

list

DEFAULT

""

GROUP

system_variables

DESCRIPTION

Leakage power can be optimized for multi-vth designs in two different flows in **place_opt**. One way is to use all of the target libraries throughout optimization. The other is to use a subset of target libraries in the first part of the optimization steps and all the libraries in the second part. The **initial_target_library** variable specifies the list of technology libraries of components to be used for the first part of leakage power optimization in **place_opt**.

SEE ALSO

```
place_opt(2)
system_variables(3)
```

insert_test_design_naming_style

Specifies how the **insert_dft** command names new designs created during the addition of test circuitry.

TYPE

string

DEFAULT

%s_test_%d

GROUP

insert_dft_variables

DESCRIPTION

This variable specifies how the **insert_dft** command names new designs created during the addition of test circuitry. When **insert_dft** modifies a design by adding test circuitry, it creates the design with a new, unique name. The new name is derived from the original design name and the format specified by this variable.

This variable must contain only one %s (percent s) and %d (percent d) character sequence. The percent sign has special meaning in the formatting process. To use a percent sign in the design name, two are needed in the variable setting (%%).

When **insert_dft** generates a new design name, it replaces %s with the original design name and %d with an integer. The integer is one that ensures the new name is unique. A single percent sign is substituted for %%.

For example, if this variable is set to *%s_test_%d*, and the original design name is *my_design*, the new design name is *my_design_test_1*.

If the **insert_dft** command is repeated (for example, with a different test methodology), the new design name is *my_design_test_2*.

To determine the current value of this variable, use the **printvar insert_test_design_naming_style** command. For a list of **insert_dft** variables and their current values, use **print_variable_group insert_dft**.

SEE ALSO

io_variables

Variables that affect the **read_file**, **read_lib**, **write**, and **write_lib** commands.

SYNTAX

```
Boolean bus_inference_descending_sort =   "true"
string bus_inference_style =      ""
string equationout_and_sign = "*"
string equationout_or_sign = "+"
Boolean equationout_postfix_negation = "true"
Boolean hdlout_internal_busses = "false"
integer libgen_max_difference = "1"
Boolean pla_read_create_flip_flop = "false"
Boolean read_db_lib_warnings = false
list read_name_mapping_nowarn_libraries = {"lsi_10k"}
Boolean read_translate_msff = false
float sdfout_min_fall_cell_delay = 0.000000
float sdfout_min_fall_net_delay = 0.000000
float sdfout_min_rise_cell_delay = 0.000000
float sdfout_min_rise_net_delay = 0.000000
float sdfout_time_scale = 1.000000
string sdfout_top_instance_name =       ""
Boolean sdfout_write_to_output =    "false"
list write_name_mapping_nowarn_libraries = {"lsi_10k"}
Boolean write_name_nets_same_as_ports = "false"
```

DESCRIPTION

These variables directly affect the **read_file**, **read_lib**, **write**, and **write_lib** commands.

For more about Verilog and VHDL variables, see the *HDL Compiler for Verilog Reference*, *VHDL Compiler Reference*, respectively, and the hdl_variables man page.

For a list of **io** variables and their current values, type **print_variable_group io**. To view this manual page online, type **help io_variables**. To view an individual variable description, type **help *var*, where *var* is the name of the variable.**

bus_inference_descending_sort

Affects the **read_file** command with all format options except **db**, **verilog**, and **vhdl**; primarily used with the **lsi** option (LSI/NDL format). When *true* (the default variable), members of the port bus will be sorted in descending order rather than in ascending order.

bus_inference_style

Affects the **read_file** command with all format options except **db**, **verilog**, and **vhdl**; primarily used with the **lsi** option (LSI/NDL format). This variable determines the style used to name inferred busses.

equationout_and_sign

Specifies the "and" sign to use when writing a design in equation format. It must be either "*" or "&". If you specify an invalid value, the default is used.

equationout_or_sign

Specifies the "or" sign to use when writing a design in equation format. It must be either "+" or " | ". If you specify and invalid value, the default is used.

equationout_postfix_negation

When *true* (the default value), the ` (apostrophe) is used as the negation operator when writing a design in equation format. When *false*, the prefix negation operator ! (exclamation point) is used.

hdlout_internal_busses

Controls how the **write -format verilog** and **write -format vhdl** commands write out bused nets by parsing their names when set to *true*.

libgen_max_difference

Specifies to **read_lib** the maximum number of differences to list between the v3.1 format description of a library cell and its statetable description. The default value, -1, allows all differences to be listed.

pla_read_create_flip_flop

Affects **read_file -f pla**. When *true*, enables output register information in PLA files to be read in and stored. When *false* (the default value), only **D** flip-flop information is read in and output register declarations are ignored.

read_db_lib_warnings

When *true*, indicates that warnings are to be printed while a technology db library is being read in with **read_file**. When *false* (the default), no warnings are given.

read_name_mapping_nowarn_libraries

Specifies a list of libraries for which no warning messages are to be issued by **read_file -f edif -names_file** if the libraries are not found. The default is to issue warning messages for all libraries not found.

read_translate_msff

When *true*, indicates that master-slave flip-flops are to be automatically translated to master-slave latches while a technology db library or a technology library is being read in.

When *false* (the default), both master and slave remain flip-flops. Note that DFT Compiler requires this variable to be set to *true*.

This variable is used while a technology db library is being read in when **read_file** is executed; and while a technology library is being read in by Library Compiler when **read_lib** is executed. The technology db library is affected ONLY if the program reports that the db library is being updated and asks you to save the results. Library Compiler always follows this variable during processing.

Note that DFT Compiler requires this variable to be set to *true*.

sdfout_min_fall_cell_delay

Specifies the minimum fall cell delay written to a timing file in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_min_fall_net_delay

Specifies the minimum fall net delay written to a timing file in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_min_rise_cell_delay

Specifies the minimum rise cell delay written to a timing file in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_min_rise_net_delay

Specifies the minimum rise net delay written to a timing file in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_time_scale

Specifies the time scale of the delays written to timing files in S.D.F. format. Delays from Design Compiler are written to timing files with **write_timing**.

sdfout_top_instance_name

Specifies the name prepended to all instance names when writing timing files in S.D.F. format. Timing files are written with the command **write_timing**. By default, **write_timing** prepends no name to all cell instance names.

sdfout_write_to_output

When *true*,**write_timing -f sdf** will write interconnect delays between cells and top level output ports, and will also write output-to-output pin IOPATH statements for cells that contain output-to-output timing arcs. The default is *false*. This variable must be set before using **write_timing**.

write_name_mapping_nowarn_libraries

Specifies a list of libraries for which no warning messages are to be issued by **write -f edif -names_file** if the libraries are not found. The default is to issue warning messages for all libraries not found.

write_name_nets_same_as_ports

When *true*, nets connected to ports have the same names as the ports in the descriptions of designs written in EDIF, LSI, or TDL format. The default is *false*. Other nets can be renamed to avoid creating shorts. Net names in the design are unchanged.

SEE ALSO

read(2)
read_lib(2)
write(2)
write_lib(2)

layer_attributes

Contains attributes related to layer.

DESCRIPTION

Contains attributes related to layer.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class layer -application**, the definition of attributes can be listed.

Layer Attributes

data_type_details

Specifies detail information of data types on a layer.

The data type of **data_type_details** is string.

This attribute is read-only.

data_types

Specifies data types on a layer.

The data type of **data_types** is string.

This attribute is read-only.

defaultWidth

Specifies the default width of any dimension of an object on a layer.

The data type of **defaultWidth** is float.

This attribute is read-only.

fatContactThreshold

Specifies the threshold for using a fat wire contact on a layer instead of the default contact.

The data type of **fatContactThreshold** is float.

This attribute is read-only.

fatFatMinSpacing

Specifies the minimum distance required between wires on a layer when the widths of both wires are greater than or equal to **fatWireThreshold**.

The data type of **fatFatMinSpacing** is float.

This attribute is read-only.

fatThinMinSpacing

Specifies the minimum distance required between wires on a layer when the width of one of the wires is greater than or equal to **fatWireThreshold**.

The data type of **fatThinMinSpacing** is float.

This attribute is read-only.

fatWireThreshold

Specifies the threshold for using the fat wire spacing rule instead of the default spacing rule on a layer.

The data type of **fatWireThreshold** is float.

This attribute is read-only.

isDefaultLayer

Specifies the layer used for routing when there are multiple layers with the same **mask_name** value.

The data type of **isDefaultLayer** is integer.

This attribute is read-only.

is_routing_layer

Specifies whether layer is a routing layer.

The data type of **is_routing_layer** is boolean.

This attribute is read-only.

layerNumber

Defines the number that identifies a layer.

The data type of **layerNumber** is integer.

This attribute is read-only.

layer_number

Defines the number that identifies a layer.

The data type of **layer_number** is integer.

This attribute is read-only.

layer_type

Specifies type of a layer.

The data type of **layer_type** is string.

This attribute is read-only.

mask_name

Specifies the physical layer associated with the specified layer object.

The data type of **mask_name** is string.

This attribute is read-only.

maxCurrDensity

Specifies the floating-point number representing in amperes per centimeter the maximum current density a layer can carry.

The data type of **maxCurrDensity** is double.

This attribute is read-only.

maxStackLevel

Defines the maximum number of vias that can stack at the same point.

The data type of **maxStackLevel** is integer.

This attribute is read-only.

minArea

Specifies the minimum area rule of any dimension of an object on a layer.

The data type of **minArea** is float.

This attribute is read-only.

minSpacing

Specifies the minimum separation distance between the edges of objects on a layer, if the objects are on different nets.

The data type of **minSpacing** is float.

This attribute is read-only.

minWidth

Specifies the minimum width of any dimension of an object on a layer.

The data type of **minWidth** is float.

This attribute is read-only.

name

Specifies name of a layer object.

The data type of **name** is string.

This attribute is read-only.

object_class

Specifies object class name of a layer, which is **layer**.

The data type of **object_class** is string.

This attribute is read-only.

pitch

Specifies the predominant separation distance between the centers of objects on a layer.

The data type of **pitch** is float.

This attribute is read-only.

preferred_direction

Specifies the preferred routing direction for a layer.

The data type of **preferred_direction** is string.

This attribute is read-only.

unitMaxCapacitance

Specifies the maximum capacitance of a layer.

The data type of **unitMaxCapacitance** is double.

This attribute is read-only.

unitMaxHeightFromSub

Specifies the maximum distance of a layer.

The data type of **unitMaxHeightFromSub** is double.

This attribute is read-only.

unitMaxResistance

Specifies the maximum resistance of a layer.

The data type of **unitMaxResistance** is double.

This attribute is read-only.

unitMaxSideWallCap

Specifies the maximum sidewall capacitance of a layer.

The data type of **unitMaxSideWallCap** is double.

This attribute is read-only.

unitMaxThickness

Specifies the maximum thickness of a layer.

The data type of **unitMaxThickness** is double.

This attribute is read-only.

unitMinCapacitance

Specifies the minimum capacitance of a layer.

The data type of **unitMinCapacitance** is double.

This attribute is read-only.

unitMinHeightFromSub

Specifies the minimum distance of a layer.

The data type of **unitMinHeightFromSub** is double.

This attribute is read-only.

unitMinResistance

Specifies the minimum resistance of a layer.

The data type of **unitMinResistance** is double.

This attribute is read-only.

unitMinSideWallCap

Specifies the minimum sidewall capacitance of a layer.

The data type of **unitMinSideWallCap** is double.

This attribute is read-only.

unitMinThickness

Specifies the minimum thickness of a layer.

The data type of **unitMinThickness** is double.

This attribute is read-only.

unitNomCapacitance

Specifies the nominal capacitance of a layer.

The data type of **unitNomCapacitance** is double.

This attribute is read-only.

unitNomHeightFromSub

Specifies the nominal distance of a layer.

The data type of **unitNomHeightFromSub** is double.

This attribute is read-only.

unitNomResistance

Specifies the nominal resistance of layer.

The data type of **unitNomResistance** is double.

This attribute is read-only.

unitNomSideWallCap

Specifies the nominal sidewall capacitance of a layer.

The data type of **unitNomSideWallCap** is double.

This attribute is read-only.

unitNomThickness

Specifies the nominal thickness of a layer.

The data type of **unitNomThickness** is double.

This attribute is read-only.

visible

Specifies a layer's visibility.

The data type of **visible** is integer.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

lbo_cells_in_regions

Puts new cells at specific locations within a cluster.

TYPE

Boolean

DEFAULT

false

GROUP

links_to_layout_variables

DESCRIPTION

Location based optimization (LBO) puts new cells at specific locations within a cluster. When this variable is set to **true**, LBO converts the specific location into a preferred region for the cell, by putting X_BOUNDS and Y_BOUNDS attributes on the cell when it is written to the PDEF file.

The proper setting for this variable depends on the engineering change order (ECO) capabilities of the back-end tools being used. Ideally the back-end tool is able to support putting the new cells exactly where the **reoptimize_design** command wants them to go. If tools do not support that level of ECO, set this variable to **true** so that the PDEF file will at least contain regions into which the cells can be placed.

To determine the current value of this variable, use the **printvar lbo_cells_in_regions** command. For a list of all **links_to_layout** variables and their current values, use **print_variable_group links_to_layout**.

SEE ALSO

legalize_enable_rpa_site_row

Specifies whether IC Compiler considers the reserved placement areas that are defined for this design during legalization. The default value is false.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

This variable specifies whether IC Compiler considers the reserved placement areas that are defined for this design during legalization. The reserved placement area site rows are the rows that are created on top of macro cells or ILM blocks. Usually, IC Compiler legalization does not place standard cells onto the rows that are on top of macro cells. If this variable is set to true and there are reserved placement area site rows in the design, IC Compiler legalization places standard cells onto the reserved placement area site rows.

Multicorner-Multimode Support

This command has no dependency on scenario-specific information.

SEE ALSO

```
check_legality(2)
legalize_placement(2)
psynopt(2)
place_opt(2)
```

legalize_use_cts_max_spacing

Controls how electromigration-aware clock tree synthesis interprets clock cell spacing rules.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls how electromigration-aware clock tree synthesis interprets the clock cell spacing rules.

By default (**false**), a pair of clock cells is separated by a distance equal to the sum of their respective spacing rules.

When set to **true**, a pair of clock cells is separated by a distance equal to the larger of their respective spacing rules.

Use the following command to determine the current value of this variable:

```
prompt> printvar legalize_use_cts_max_spacing
```

EXAMPLES

In the following example, cells of CLK1 will be 0.25um apart in the x-direction and 0.25um apart in the y-direction:

```
prompt> set_clock_cell_spacing -clock_trees CLK1 \
   -x_spacing 0.25 -y_spacing 0.25
prompt> set legalize_use_cts_max_spacing true
```

In the following example, cells of CLK1 will be 0.5um apart in the x-direction and 0.5um apart in the y-direction:

```
prompt> set_clock_cell_spacing -clock_trees CLK1 \
```

```
   -x_spacing 0.25 -y_spacing 0.25
prompt> set legalize_use_cts_max_spacing false
```

SEE ALSO

```
set_clock_cell_spacing(2)
remove_clock_cell_spacing(2)
report_clock_cell_spacing(2)
```

legalizer_consider_vth_spacing

Enables legalization, legality checking, and filler cell insertion to consider the threshold voltage rules.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable enables the threshold voltage rules for legalization, legality checking, and filler insertion.

The legalizer considers the minimum width and minimum spacing rules of the threshold voltage implant layers and adjusts the cell placement to meet these rules.

The **check_legality** command reports violations of these rules.

Filler cell insertion uses the threshold voltage rules to determine the proper filler cells to insert.

By default (**false\P), the threshold voltage rules are not considered. Setting this variable to true might impact QoR and legalization runtime.**

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_consider_vth_spacing
```

SEE ALSO

```
legalize_placement(2)
check_legality(2)
insert_stdcell_filler(2)
```

legalizer_enable_via_spacing_check

Enables the via spacing violation checks performed by the legalizer.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable allows you to enable the legalizer checks for spacing violations on the via cut layer, between vias embedded within reference cells and Power and Ground vias of the power mesh, as defined in the Technology file. By default, via layer spacing checks are not performed. Checking for all via cut layers is enabled with this variable. Setting this variable might impact QoR and legalization runtime.

The default is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_enable_via_spacing_check
```

SEE ALSO

```
legalize_placement(2)
check_legality(2)
```

legalizer_skip_preroute_merge

Controls whether the initialization stage of the legalizer uses the preroute merge algorithm.

TYPE

Boolean

DEFAULT

true

GROUP

placement

DESCRIPTION

This variable controls whether the initialization stage of the legalizer uses the preroute merge algorithm.

The preroute merge algorithm combines preroutes that are separated by a distance equivalent to N tracks into a single preroute for legalization checking. Preroutes on all metal layers are considered for merging if the Preroute merge algorithm is used (e.g. metal2 preroutes are merged with metal2 preroutes; metal3 preroutes are merged with metal3 preroutes, etc.) By default, N is 1. You can change the value of N (integer) by setting variable "legalizer_preroute_merge_num_tracks" to N.

By default, the legalizer does not use the preroute merge algorithm.

To use the preroute merge algorithm, set this variable to **false**.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_skip_preroute_merge
```

EXAMPLE

In this example, N is set to two and the preroute merge algorithm is enabled.

```
prompt> set legalizer_preroute_merge_num_tracks 2
prompt> set legalizer_skip_preroute_merge false
```

SEE ALSO

legalize_placement(2)

legalizer_skip_via_access_check

Sets the variable legalizer_skip_via_access_check.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable allows the user to skip access checking during legalization. This may be useful for certain floorplans. Note that this variable turns off access checking for pins on all metal routing layers. Optionally, the user may choose to skip the access check only on the specified layer with the variables "legalizer_skip_via_access_check_of_M<layer>".

The default value of legalizer_skip_via_access_check is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_skip_via_access_check
```

EXAMPLE

In this example the user sets pnet checking to -partial on metal2..metal4 and turns off access checking on all layers.

```
prompt> set_pnet_options -partial {metal2 metal3 metal4}
prompt> set legalizer_skip_via_access_check true
```

SEE ALSO

```
legalize_placement(2)
set_pnet_options(2)
report_pnet_options(2)
legalizer_skip_via_access_check_of_M1(3)
```

legalizer_skip_via_access_check_of_M1

Controls whether the legalizer skips access checking for pins on the metal1 layer.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable controls whether the legalizer skips access checking for pins on the metal1 layer. This might be useful for certain floorplans. Note that this variable turns off checking only on the metal1 layer. Pins on other layers are still subject to access checking. To skip access checking for all layers, set the **legalizer_skip_via_access_check** variable to **true**.

The default for the **legalizer_skip_via_access_check_of_M1** variable is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_skip_via_access_check_of_M1
```

EXAMPLE

The following example disables access checking on the metal1 layer.

```
prompt> set_app_var legalizer_skip_via_access_check_of_M1 true
```

SEE ALSO

```
legalize_placement(2)
legalizer_skip_via_access_check(3)
```

legalizer_skip_via_access_check_of_M2

Controls whether the legalizer skips access checking for pins on the metal2 layer.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable controls whether the legalizer skips access checking for pins on the metal2 layer. This might be useful for certain floorplans. Note that this variable turns off checking only on the metal2 layer. Pins on other layers are still subject to access checking. To skip access checking for all layers, set the **legalizer_skip_via_access_check** variable to **true**.

The default for the **legalizer_skip_via_access_check_of_M2** variable is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_skip_via_access_check_of_M2
```

EXAMPLE

The following example disables access checking on the metal2 layer.

```
prompt> set_app_var legalizer_skip_via_access_check_of_M2 true
```

SEE ALSO

```
legalize_placement(2)
legalizer_skip_via_access_check(3)
```

legalizer_skip_via_access_check_of_M3

Controls whether the legalizer skips access checking for pins on the metal3 layer.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable controls whether the legalizer skips access checking for pins on the metal3 layer. This might be useful for certain floorplans. Note that this variable turns off checking only on the metal3 layer. Pins on other layers are still subject to access checking. To skip access checking for all layers, set the **legalizer_skip_via_access_check** variable to **true**.

The default for the **legalizer_skip_via_access_check_of_M3** variable is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_skip_via_access_check_of_M3
```

EXAMPLE

The following example disables access checking on the metal3 layer.

```
prompt> set_app_var legalizer_skip_via_access_check_of_M3 true
```

SEE ALSO

```
legalize_placement(2)
legalizer_skip_via_access_check(3)
```

legalizer_skip_via_access_check_of_M4

Controls whether the legalizer skips access checking for pins on the metal4 layer.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable controls whether the legalizer skips access checking for pins on the metal4 layer. This might be useful for certain floorplans. Note that this variable turns off checking only on the metal4 layer. Pins on other layers are still subject to access checking. To skip access checking for all layers, set the **legalizer_skip_via_access_check** variable to **true**.

The default for the **legalizer_skip_via_access_check_of_M4** variable is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_skip_via_access_check_of_M4
```

EXAMPLE

The following example disables access checking on the metal4 layer.

```
prompt> set_app_var legalizer_skip_via_access_check_of_M4 true
```

SEE ALSO

```
legalize_placement(2)
legalizer_skip_via_access_check(3)
```

legalizer_skip_via_access_check_of_M5

Controls whether the legalizer skips access checking for pins on the metal5 layer.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable controls whether the legalizer skips access checking for pins on the metal5 layer. This might be useful for certain floorplans. Note that this variable turns off checking only on the metal5 layer. Pins on other layers are still subject to access checking. To skip access checking for all layers, set the **legalizer_skip_via_access_check** variable to **true**.

The default for the **legalizer_skip_via_access_check_of_M5** variable is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_skip_via_access_check_of_M5
```

EXAMPLE

The following example disables access checking on the metal5 layer.

```
prompt> set_app_var legalizer_skip_via_access_check_of_M5 true
```

SEE ALSO

```
legalize_placement(2)
legalizer_skip_via_access_check(3)
```

legalizer_skip_via_access_check_of_M6

Controls whether the legalizer skips access checking for pins on the metal6 layer.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable controls whether the legalizer skips access checking for pins on the metal6 layer. This might be useful for certain floorplans. Note that this variable turns off checking only on the metal6 layer. Pins on other layers are still subject to access checking. To skip access checking for all layers, set the **legalizer_skip_via_access_check** variable to **true**.

The default for the **legalizer_skip_via_access_check_of_M6** variable is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_skip_via_access_check_of_M6
```

EXAMPLE

The following example disables access checking on the metal6 layer.

```
prompt> set_app_var legalizer_skip_via_access_check_of_M6 true
```

SEE ALSO

```
legalize_placement(2)
legalizer_skip_via_access_check(3)
```

legalizer_unit_tile_names

Specifies the names of the unit tiles in which to perform filler insertion or tap insertion when the design contains rows of different unit tiles.

TYPE

string

DEFAULT

""

GROUP

placement

DESCRIPTION

Filler insertion and tap insertion can only work on rows of same unit tile. If design contains rows of different unit tiles, use this variable to specify the unit tiles of the rows in which to perform filler insertion or tap insertion. If this variable is not set, the results might not be correct.

For example, the following script performs fill insertion on rows containing "unit1" unit tiles first, then performs fill insertion on rows containing "unit2" unit tiles.

```
set_app_var legalizer_unit_tile_names "unit1"
insert_stdcell_filler ...
set_app_var legalizer_unit_tile_names "unit2"
insert_stdcell_filler ...
```

By default, this variable is set to an empty string("") and tool uses the first unit tile it reads in.

SEE ALSO

```
add_tap_cell_array(2)
insert_stdcell_filler(2)
```

legalizer_use_pin_via

Sets the variable legalizer_use_pin_via.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

This variable allows the user to enable the legalizer to use the via associated with each reference pin for legalization checking of the pin. By default, a representative via is used for all pins in the design. In certain testcases the default methodology may be problematic if different pins are associated with vias of different sizes and thus the legalizer may not produce correct results. Note, small degradation in QoR is possible if this variable is used.

The default value of legalizer_use_pin_via is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar legalizer_use_pin_via
```

SEE ALSO

```
legalize_placement(2)
```

level_shifter_naming_prefix

Specifies a prefix for level shifter names.

TYPE

string

DEFAULT

""

GROUP

mv

DESCRIPTION

When the **enable_special_level_shifter_naming** variable is set to **true**, level shifters that are automatically inserted by the **insert_level_shifters**, **compile** and other commands are specially named. The name follows the template <prefix> + <PD OR Design name> + "_LS" + #. The <prefix> is a user-specified prefix using the **level_shifter_naming_prefix** variable. The <PD OR Design Name> is the name of power domain where the level shifter is being added, or the design name if the power domain is not defined

SEE ALSO

`enable_special_level_shifter_naming(3)`

lib_cell_using_delay_from_ccs

Controls whether the tool uses information from CCS tables instead of NLDM library timing information for cases with mixed CCS and NLDM libraries.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** and the library contains both CCS timing and NLDM timing, the tool replaces the NLDM library timing with the timing information derived from CCS. When the variable is set to **false** and the library contains both CCS timing and NLDM timing, the tool uses the NLDM library timing from library NLDM data. This variable only works when it is set before library loading.

For the current value of this variable, use the **printvar lib_cell_using_delay_from_ccs** command.

SEE ALSO

`lib_pin_using_cap_from_ccs(3)`

lib_pin_using_cap_from_ccs

Controls whether the tool uses the pin caps derived from CCS in cases with mixed CCS and NLDM libraries.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** and the library contains both CCS timing and NLDM timing, the tool uses the pin caps derived from CCS. When the variable is set to **false** and the library contains both CCS timing and NLDM timing, the tool uses the NLDM library pin cap. This variable only works when it is set before library loading.

For the current value of this variable, use the **printvar lib_pin_using_cap_from_ccs** command.

SEE ALSO

`lib_cell_using_delay_from_ccs(3)`

lib_use_thresholds_per_pin

Causes pin-specific trip-point values in the Synopsys library to override library default trip-point values.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

Setting this variable to **false** causes Synopsys Library default trip-point values to override values defined for each library pin in the Synopsys library.

This variable is provided for backward compatibility. This variable allows you to use library defaults for all library pins instead of pin specific trip_point values.

The following variables are affected:

```
lib_thresholds_per_lib
rc_input_threshold_pct_rise
rc_input_threshold_pct_fall
rc_output_threshold_pct_rise
rc_output_threshold_pct_fall
rc_slew_derate_from_library
rc_slew_lower_threshold_pct_fall
rc_slew_lower_threshold_pct_rise
rc_slew_upper_threshold_pct_fall
rc_slew_upper_threshold_pct_rise
```

To determine the current value of this variable, use the **printvar lib_use_thresholds_per_pin** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

SEE ALSO

rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_derate_from_library(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)

libgen_max_differences

Specifies to the **read_lib** command the maximum number of differences to list between the v3.1 format description of a library cell and its statetable description.

TYPE

integer

DEFAULT

-1

GROUP

io_variables

DESCRIPTION

Specifies to the **read_lib** command the maximum number of differences to list between the v3.1 format description of a library cell and its statetable description. The default value of -1 allows all differences to be listed.

For example, if **libgen_max_differences** = 5, **read_lib** lists only up to 5 differences between a library cell's v3.1 format description and its statetable description.

To see the value of this variable, type **printvar libgen_max_differences**. For a list of all **io** variables and their values, type **print_variable_group io**.

SEE ALSO

```
read_lib(2)
io_variables(3)
```

library_attributes

Contains attributes placed on libraries.

DESCRIPTION

Contains attributes that can be placed on a library.

To set library attributes, use the **set_attribute** command. To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For more details on library attributes, see the *Library Compiler Reference Manual*. For information on all attributes, refer to the **attributes** manual page.

Library Attributes

default_fanout_load

Fanout load of input pins in a library.

default_inout_pin_cap

Capacitance of inout pins.

default_inout_pin_fall_res

Fall resistance of inout pins.

default_inout_pin_rise_res

Rise resistance of inout pins.

default_input_pin_cap

Capacitance of input pins.

default_intrinsic_fall

Intrinsic fall delay of timing arcs.

default_intrinsic_rise

Intrinsic rise delay of a timing arc.

default_max_fanout

Maximum fanout of pins.

default_max_transition

Maximum transition of pins.

default_min_porosity

Minimum porosity of designs.

default_output_pin_cap

Capacitance of output pins.

default_output_pin_fall_res

Fall resistance of output pins.

default_output_pin_rise_res

Rise resistance of output pins.

default_slope_fall

Fall sensitivity factor of a timing arc.

default_slope_rise

Rise sensitivity factor of a timing arc.

k_process_drive_fall

Process scale factor applied to the fall resistance of timing arcs.

k_process_drive_rise

Process scale factor applied to the rise resistance of timing arcs.

k_process_intrinsic_fall

Process scale factor applied to the intrinsic fall delay of timing arcs.

k_process_intrinsic_rise

Process scale factor applied to the intrinsic rise delay of timing arcs.

k_process_pin_cap

Process scale factor applied to pin capacitance of timing arcs.

k_process_slope_fall

Process scale factor applied to the fall slope sensitivity of timing arcs.

k_process_slope_rise

Process scale factor applied to the rise slope sensitivity of timing arcs.

k_process_wire_cap

Process scale factor applied to the wire capacitance of timing arcs.

k_process_wire_res

Process scale factor applied to the wire resistance of timing arcs.

k_temp_drive_fall

Scale factor applied to timing arc fall resistance due to temperature variation.

k_temp_drive_rise

Scale factor applied to timing arc rise resistance due to temperature variation.

k_temp_intrinsic_fall

Scale factor applied to the intrinsic fall delay of a timing arc due to temperature variation.

k_temp_intrinsic_rise

Scale factor applied to the intrinsic rise delay of a timing arc due to temperature variation.

k_temp_pin_cap

Scale factor applied to pin capacitance due to temperature variation.

k_temp_slope_fall

Scale factor applied to timing arc fall slope sensitivity due to temperature variation.

k_temp_slope_rise

Scale factor applied to timing arc rise slope sensitivity due to temperature variation.

k_temp_wire_cap

Scale factor applied to wire capacitance due to temperature variation.

k_temp_wire_res

Scale factor applied to wire resistance due to temperature variation.

k_volt_drive_fall

Scale factor applied to timing arc fall resistance due to voltage variation.

k_volt_drive_rise

Scale factor applied to timing arc rise resistance due to voltage variation.

k_volt_intrinsic_fall

Scale factor applied to the intrinsic fall delay of a timing arc due to voltage variation.

k_volt_intrinsic_rise

Scale factor applied to the intrinsic rise delay of a timing arc due to voltage variation.

k_volt_pin_cap

Scale factor applied to pin capacitance due to voltage variation.

k_volt_slope_fall

Scale factor applied to timing arc fall slope sensitivity due to voltage variation.

k_volt_slope_rise

Scale factor applied to timing arc rise slope sensitivity due to voltage variation.

k_volt_wire_cap

Scale factor applied to wire capacitance due to voltage variation.

k_volt_wire_res

Scale factor applied to wire resistance due to voltage variation.

nom_process

Nominal process value used for library characterization. Fixed at 1.0 for most technology libraries.

nom_temperature

Nominal ambient temperature used for library characterization. Usually 25 degrees Celsius. Multipliers use the nominal value to determine the change in temperature between nominal and operating conditions.

nom_voltage

Nominal source voltage value used in library element characterization. Typically 5 volts for a CMOS library. Multipliers use the nominal value to determine the change in voltage between nominal and operating conditions.

no_sequential_degenerates

When *true*, disables mapping to degenerated flip-flops or latches (that is, devices that have some input pins connected to 0 or to 1). The default for the attribute is *false\P or nonexistence, implying that degenerate devices will be allowed by default in the library. This attribute may be overridden on a component-by-component basis by using the attribute on library cells.*

sequential_bridging

When *true*, enables **Design Compiler** to take a multiplexed flip-flop and bridge (that is, connect) the output to the input to get a desired functionality. The default for the attribute is

false or nonexistent, implying that bridging will be disabled by default in the library. Bridging is required for mapping in cases where there is no flip-flop with internal feedback in the target library but one is desired in the HDL. This attribute may be overridden on a component-by-component basis by using the attribute on library cells.

NOTE: Setting this attribute to *true* can result in an increase in run times and memory consumption for Design Compiler. The increased run times depend on the number of flip-flops in the target library or libraries which have the attribute set.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)
```

library_cell_attributes

Contains attributes that can be placed on a library cell.

DESCRIPTION

Contains attributes that can be placed on a library cell.

To set an attribute, use the command identified in the individual description of that attribute. To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command. If an attribute is "read-only," you cannot set it.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Library Cell Attributes

dont_touch

Identifies library cells to be excluded from optimization. Values are *true* (the default) or *false*. Library cells with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Setting **dont_touch** on a hierarchical cell sets the attribute on all cells below it. Set with **set_dont_touch**.

dont_use

Disables the specified library cells so that they are not added to a design during **compile**. Set with **set_dont_use**.

no_sequential_degenerates

When *true*, disables mapping to versions of this latch or flip flop that have some input pins connected to 0 or to 1. Set with **set_attribute**. This attribute may also be set on the library itself, and that value will apply as the default for all registers in the library which do not have the attribute set individually.

preferred

Specifies the preferred library gate to use during technology translation when there are other gates with the same function in the target library. Set with **set_prefer**.

scan

When *true*, specifies that the instances of the library cell are always replaced by equivalent scan cells during **insert_dft**. When *false*, instances are not replaced. Set with **set_scan**.

sequential_bridging

When *true*, enables Design Compiler to take a multiplexed flip-flop and bridge (that is, connect) the output to the input to get a desired functionality. The default is *false*, so this attribute must be set in order to enable the functionality. Bridging is required for mapping in cases where there is no flip-flop with internal feedback in the target library but one is desired in the HDL. Set with **set_attribute**. This attribute may also be set on the library itself, and that value will apply as the default for all registers in the library which do not have the attribute set individually.

NOTE: Setting this attribute to *true* can result in an increase in run times and memory consumption for Design Compiler. The increased run times depend on the number of flip-flops in the target library or libraries for which this attribute has been set.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)
```

link_allow_design_mismatch

Allows the tool to link and proceed with the flow even if some design mismatches exist.

TYPE

Boolean

DEFAULT

false

GROUP

system_variables

DESCRIPTION

When set to **true**, this variable allows the tool to link and proceed with the flow even if some design mismatches exist.

To determine the current setting of this variable, use the **printvar link_allow_design_mismatch** command. For a list of all system variables and their current settings, use the **print_variable_group** system command.

SEE ALSO

`report_design_mismatch(2)`

link_force_case

Controls the case-sensitive or case-insensitive behavior of the **link** command.

TYPE

string

DEFAULT

check_reference

GROUP

system_variables

DESCRIPTION

Controls the case-sensitive or case-insensitive matching policy to be used on HDL identifiers considered during a **link** command. The value of this variable can be set to **check_reference** (the default), **case_sensitive**, or **case_insensitive**.

By default, the reference and the design cells to be linked are checked to ascertain the alphabet-case matching policy of the HDL input format that created that particular cell. Then, the least-sensitive matching rule is applied to determine whether the design cell resolves the reference. For example, a VHDL reference is linked case-insensitively and a Verilog reference is linked case-sensitively, while mixed language linkage candidates match case-insensitively. To override this behavior, you can set the value of the **link_force_case** variable to either **case_sensitive** or **case_insensitive**.

The appropriate matching policy applies to all the HDL identifiers that must be matched by the link: design template, port, and parameter names. It does not apply to the (canonical forms of) data values in any parameter override, which (for string data) are always case-sensitive. It also does not apply to library and architecture identifiers, which are always case-insensitive. When link candidates are built by elaboration, case-forcing does not change the HDL identifiers of the design or reference cells that are built.

Setting this variable to either the **case_sensitive** or the **case_insensitive** value can lead to results which are inconsistent with the default rules, since a forced rule applies throughout the linked hierarchy.

To determine the value of this variable, use the **printvar link_force_case** command.

SEE ALSO

`link(2)`

link_library

Specifies the list of design files and libraries used during linking.

TYPE

list

DEFAULT

* your_library.db

GROUP

system_variables

DESCRIPTION

This variable specifies the list of design files and libraries used during linking. The **link** command looks at the files and tries to resolve references in the order of the specified files. A "*" entry in the value of this variable indicates that the **link** command is to search all the designs loaded in dc_shell while trying to resolve references. If file names do not include directory names, files are searched for in the directories in **search_path**. The default is {"*" your_library.db}. Change *your_library.db* to reflect your library name.

To determine the current value of this variable, use the **printvar link_library** command. For a list of all system variables and their current values, use the **print_variable_group system** command.

SEE ALSO

`link(2)`

ltl_obstruction_type

Controls the routing blockage type for the named obstructions, without the route type being specified.

TYPE

Boolean

DEFAULT

placement_only

GROUP

links_to_layout_variables

DESCRIPTION

This variable controls the routing blockage type for the named obstructions, without the route type being specified. When the setting is **placement_only** (the default), the obstructions are treated as placement obstructions only and routing wires can still go through. When the setting is **routing_none**, the obstructions are treated as routing blockages and no routing wires are allowed.

To determine the current value of this variable, use the **printvar ltl_obstruction_type** command. For a list of all **links_to_layout** variables and their current values, use **print_variable_group links_to_layout**.

SEE ALSO

`lbo_cells_in_regions(3)`

magnet_placement_disable_overlap

Controls whether the **magnet_placement** command can move pre-placed cells during magnet placement.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the **magnet_placement** command moves the available pre-placed cells in the chip area during magnet placement.

If the **magnet_placement_disable_overlap** variable is set to true, the location of pre-placed cells is kept. The **magnet_placement** command looks for other legal locations to place magnet objects. If no legal location is found, the magnet objects are not pulled closer to the specified magnet.

SEE ALSO

```
get_magnet_cells(2)
magnet_placement(2)
```

magnet_placement_fanout_limit

Sets the threshold of the high-fanout nets during the **magnet_placement** command.

TYPE

integer

DEFAULT

1000

DESCRIPTION

The **magnet_placement_fanout_limit** variable controls whether the nets with high fanout are pulled during magnet placement. If the net has more fanouts than the specified threshold value, the cells on the net are not pulled.

SEE ALSO

```
get_magnet_cells(2)
magnet_placement(2)
```

magnet_placement_stop_after_seq_cell

Controls whether sequential cells are pulled towards a specified magnet during magnet placement.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether sequential cells are pulled towards a specified magnet during magnet placement.

When **false** (the default), sequential cells are not pulled towards a specified magnet when you run the **magnet_placement** command with the **-stop_by_sequential_cells** option.

When **true**, sequential cells are pulled towards a specified magnet when you run the **magnet_placement** command with the **-stop_by_sequential_cells** option.

By default, the magnet placement operation terminates before sequential cells when you run the **magnet_placement** command with the **-stop_by_sequential_cells** option.

SEE ALSO

```
get_magnet_cells(2)
magnet_placement(2)
```

mcmm_high_capacity_effort_level

Controls the behavior of multicorner-multimode (MCMM) scenario reduction.

TYPE

Float. Valid range is between 0 and 10.

DEFAULT

0

GROUP

MCMM

DESCRIPTION

This variable controls how aggressively the multicorner-multimode scenario reduction feature reduces the number of dominant scenarios.

By default, any scenario with a violation that is worst across all scenarios is included in the dominant set of scenarios. As you increase the value of this variable, scenario reduction adjusts its criteria for comparing the slack of a violating object across different scenarios. This allows for further reduction of the dominant scenario set, but implies that a few violations will not be fixed during optimization.

If you set this variable to a value smaller than 0, scenario reduction is performed using an effort level of 0. If you set this variable to a value larger than 10, scenario reduction is performed using an effort level of 10.

See the man page for the **get_dominant_scenarios** command for more information about scenario reduction.

SEE ALSO

```
all_active_scenarios(2)
all_scenarios(2)
get_dominant_scenarios(2)
place_opt(2)
```

```
route_opt(2)
set_active_scenarios(2)
```

mmcts_scenario_order

Specifies the ordering of multicorner-multimode scenarios used for multimode clock tree synthesis (MMCTS)and multimode clock tree optimization (MMCTO)

TYPE

list

DEFAULT

""

GROUP

cts_variables

DESCRIPTION

This variable specifies the scenario ordering for MMCTS and MMCTO. When this variable is set with the scenario names, MMCTS and MMCTO works on the clocks scenario-by-scenario, based on the ordering of this scenario list specified. In this way, you can control MMCTS or MMCTO to synthesize or optimize the clocks of the more important scenarios before the less important scenarios. You should only specify the scenarios which have the cts_mode set to true. If a scenario which has the cts_mode set to false is included, it is ignored. If you specify only some, but not all, of the scenarios with the cts_mode true, MMCTS and MMCTO honors only those scenarios, in the order given. The clocks of the scenarios with cts_mode true that are not in this list follows an internally-determined order, and might appear before or after the clocks of those listed scenarios. This variable only controls the ordering of the master clocks, but not that of the generated clocks.

SEE ALSO

`compile_clock_tree(2)`

monitor_cpu_memory

Displays the CPU time, elapsed time, and peak memory usage before and after each core command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When **monitor_cpu_memory** is true, the CPU time, elapsed time, and peak memory usage are printed out before and after each major command, such as **place_opt** and **clock_opt**.

CPU time and elapsed time are the total time spent on the main process and all its child processes or multiple threads. For example, when you use the **-num_cpus** option, the reported CPU time includes the sum of the CPU usage of the main process and all child processes or threads, similar to the output of the **cputime -self -child** or **cputime -all** command.

Peak memory is defined as the memory high-water mark of the main process and its child processes.

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

When you use the **-num_cpus** option, the tool runs multiple threads. Currently the operating system measures the CPU time by adding the usage for all threads. It does not take into account parallelization of the threads. This means that the report might show a larger CPU time than elapsed time when you specify the **-num_cpus** option.

The elapsed time (wall-clock time) depends on many external factors and can vary for every session. It depends on factors such as the I/O traffic, the network traffic, RAM and swap usage, and the other processes running on the same machine. When the elapsed time is much longer than the CPU time, it might point out an inefficiency of the computing environment, such as insufficient memory, too many processes running on the same machine, or a slow network. The only way to make the elapsed time close to the CPU time is to run only one session on a machine with enough RAM and using the local disk.

EXAMPLE

If set the **monitor_cpu_memory** variable to true, the tool outputs a PSYN-508 message in the log file after each major command. A sample PSYN-508 message is shown below:

Information: CPU: ### s (## hr) ELAPSE: ### s (## hr) MEM-PEAK: ### mb Mon Oct 28 12:20:01 2008 (PSYN-508)

SEE ALSO

`mem(2)`
`cputime(2)`
`PSYN-508(n)`

mux_auto_inferring_effort

Specifies the MUX inferring effort level.

TYPE

integer

DEFAULT

2

GROUP

fpga_variables

DESCRIPTION

The **mux_auto_inferring_effort** variable controls the MUX inferring effort of Design Compiler FPGA. Valid values are **0** through **6**. The default value is **2**. The larger the integer value, the more MUX is inferred.

mv_allow_ls_on_leaf_pin_boundary

Allows level-shifter insertion on leaf pin (such as macro cell pin) boundaries.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls whether or not to allow level-shifter insertion on leaf pin boundaries that are not power domain boundaries. When a macro cell is operating at a voltage different from its surrounding logic, and you want level shifters to be inserted at the interface, the recommended flow is to define a power domain around the macro cell by specifying the macro cell as the root cell of a power domain.

If you do not define the macro cell as the root cell, level shifters are not inserted at the interface, because the interface is not a power domain boundary. By default, level shifters are only inserted at power domain boundaries.

If you are unable to define a power domain around the macro cell, but still require level shifters to be inserted, use this variable to enable the non-default behavior.

SEE ALSO

mv_allow_pg_pin_reconnection

Allows command connect_supply_net to reconnect pg pins.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls whether or not to allow reconnection on PG pins. By default, the connect_supply_net command errors out when connecting a supply net to a PG pin, which already has a supply net connection.

By setting this variable to true, the connect_supply_net command reconnects the new supply net to the PG pin and the previous connection is discarded.

SEE ALSO

`connect_supply_net(2)`

mv_continue_on_opcond_mismatch

Sets the variable to **true** to continue optimization process when there is an operating condition mismatch between design constraint and the library cell and the tool has issued LIBSETUP-001 error message.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls the continuation of the optimization process when there is an operating condition mismatch between design constraint and the library, and LIBSETUP-001 and LIBSETUP-022 error messages are issued.

By default, the optimization stops when the LIBSETUP-001 error message is issued. Check the details of the LIBSETUP-001 error message and fix it before continuing with the optimization.

Do not set this variable to continue the flow unless you are sure that the operating condition mismatch is expected and does not cause unexpected results.

SEE ALSO

```
LIBSETUP-001(n)
LIBSETUP-022(n)
```

mv_disable_voltage_area_aware_detour_routing

This variable determines whether virtual routes used in analysis and optimization need to consider voltage areas as blockages. By default, a net from one voltage area (or the default voltage area) needs to detour around other voltage areas. Set this variable to **true** to ignore the presence of voltage areas.

You can control the behavior of the router toward voltage areas by using the **-enforce_voltage_areas** option of the **set_route_zrt_common_options** command, or set this property for specific nets by using the **-ignore_voltage_areas** option of the **set_zrt_net_properties** command.

TYPE

Boolean

DEFAULT

false

SEE ALSO

```
set_route_zrt_common_options(2)
set_zrt_net_properties(2)
```

mv_input_enforce_simple_names

Enforces the use of simple names for restricted commands as per the IEEE 1801 (UPF) standard.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls the use of hierarchical names for the arguments of certain restricted commands that require simple names as per the IEEE 1801 (UPF) standard.

The default value of this variable is false. So, by default, the tool accepts hierarchical names, even though the IEEE 1801 (UPF) standard requires them to be simple.

When you set this variable to true, the tool errors out when you use hierarchical names for the arguments of certain restricted commands. This variable is honored when you specify the inputs in ASCII, UPF, and .ddc file formats.

SEE ALSO

`mv_output_enforce_simple_names(3)`

mv_insert_level_shifters_on_ideal_nets

Directs automatic level-shifter insertion to insert level shifters on ideal nets.

TYPE

string

DEFAULT

""

GROUP

none

DESCRIPTION

When this variable is set to "all", automatic level-shifter insertion inserts level shifters on all ideal nets that need level shifters.

By default, automatic level-shifter insertion does not insert level shifters on ideal nets.

Note that if a net is also a clock net, automatic level-shifter insertion does not insert any level shifter on it, unless the variable **auto_insert_level_shifters_on_clocks** is set to "all" or contains the net name.

SEE ALSO

```
auto_insert_level_shifters(3)
auto_insert_level_shifters_on_clocks(3)
create_clock(2)
set_ideal_network(2)
```

mv_make_primary_supply_available_for_always_on

Allows the primary supply of a power domain to be used for buffering always-on feedthrough nets passing through the domain.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

This variable determines whether always-on synthesis can use the primary supplies of a power domain for buffering feedthrough nets that pass through the domain.

By default, the variable is set to **true**, which allows the primary supplies of a power domain to be used for buffering always-on feedthrough nets, as long as doing so does not introduce electrical violations. Buffers ordinarily used in the power domain are used as much as possible to get the best possible QoR.

If the variable is set to **false**, feedthrough nets are marked as always-on nets with the related supply from driver or loads, requiring that any buffers used on the feedthrough net be compatible with the driver or load domains. This option is provided mainly for backward compatibility with previous releases of the tool, which always had this requirement.

SEE ALSO

```
get_always_on_logic(2)
analyze_mv_design(2)
```

mv_mtcmos_detour_obstruction

Allows the connection of power switch cells to bypass obstructions.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls whether or not to detour obstructions such as blockages and macro cells when connecting power switch cells.

This variable setting is applicable only when you use the **connect_power_switch** command with the **-mode fishbone** option.

SEE ALSO

`connect_power_switch(2)`

mv_no_always_on_buffer_for_redundant_isolation

Allows normal buffers to be used at nets driving the data input of redundant isolation cells.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls whether or not always-on buffers are used on the nets driving the data-input pins of redundant isolation cells. If the variable is set to **true**, always-on buffer or inverter cells are avoided on these nets in favor of regular buffer or inverter cells supplied by the local power of the net's power domain. By default, any necessary buffering is done using the global driver supply of the net.

The net segment between domain boundary and data-input pin of redundant isolation cell will still be dont_touched and not allowed for buffering.

SEE ALSO

```
set_isolation(2)
set_isolation_control(2)
```

mv_no_cells_at_default_va

Controls whether the tool can place new buffers at the default voltage area.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable, when set to **false** (the default), allows the tool to place new buffers at the default voltage area.

When this variable is set to **true**, the buffers created during optimization are not allowed to be placed at the default voltage area.

Set this variable to **true** for a physically abutted design where there is no space in the default voltage area, or in other situations to prevent the tool from inserting new buffers at the default voltage area.

This variable is honored by the **place_opt**, **clock_opt**, and **route_opt** commands.

SEE ALSO

```
place_opt(2)
clock_opt(2)
route_opt(2)
```

mv_no_main_power_violations

Selects Standard Cell Main Rail (SCMR) as the main power supply for level shifters.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

This variable controls whether or not level shifters use SCMR as their main power supply. By default, level shifters must get their main power supply from SCMR.

To allow the tool to select other power sources as the main power supply for level shifters, set this variable to **false**.

SEE ALSO

```
check_mv_design(2)
insert_level_shifters(2)
insert_mv_cells(2)
```

mv_output_enforce_simple_names

Enforces the use of simple names for restricted commands as per the IEEE 1801 (UPF) standard.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls the printing of hierarchical names for the arguments of certain restricted commands in the UPF file or on the standard output device. The IEEE 1801 standard requires that the arguments of these commands accept only simple names.

The default value of this variable is false. So, by default, hierarchical names are printed, although the IEEE 1801 (UPF) standard requires them to be simple.

When you set this variable to true, these restricted commands print only simple names for the arguments in the UPF file or on the standard output. The tool inserts the **set_scope** command appropriately to set the current scope at the proper hierarchy. This ensures that the use of a simple name is sufficient. After printing, the original scope is returned.

SEE ALSO

`mv_input_enforce_simple_names(3)`

mv_output_upf_line_indent

Sets the number of indentation spaces inserted at the beginning of each line when **save_upf** splits long commands onto multiple lines.

TYPE

integer

DEFAULT

2

DESCRIPTION

When line splitting for the **save_upf** command is enabled, this variable specifies the number of spaces added to the beginning of each line written, except for the first line of each command (which is not indented).

Line splitting for the **save_upf** command is controlled by the **mv_output_upf_line_width** variable.

When line splitting for the **save_upf** command is disabled, the **mv_output_upf_line_indent** variable has no effect.

SEE ALSO

```
save_upf(2)
mv_output_upf_line_width(3)
```

mv_output_upf_line_width

Specifies whether the **save_upf** command writes out long commands as multiple lines, and if so, the threshold for splitting lines.

TYPE

Integer

DEFAULT

0

DESCRIPTION

When this variable is set to 0, line splitting for **save_upf** is turned off. The command writes out each UPF command as a single line, regardless of its length.

If this variable is set to a positive value, every command that is longer than that threshold is written out in multiple lines, each line containing no more than the specified number of characters, and linked by the backslash character at the end of each continuing line. You cannot set this variable to a negative number.

The second and subsequent lines of a UPF command are indented (offset to the right) by the number of spaces determined by the **mv_output_upf_line_indent** variable.

SEE ALSO

```
save_upf(2)
mv_output_upf_line_indent(3)
```

mv_skip_opcond_checking_for_unloaded_level_shifter

Skips operating condition checking for level-shifter cells with unconnected output pins.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls the checking of operating condition on level-shifter cells with unconnected output pins.

By default, operating condition checking is performed on all level-shifter cells in the design.

For the tool to skip operating condition checking for level-shifter cells with unconnected output pins, set this variable to **true**.

SEE ALSO

```
check_mv_design(2)
insert_level_shifters(2)
insert_mv_cells(2)
set_opcond_inference(2)
```

mv_upf_enable_flipped_bias_connection

Enables connections between power nets and pwell bias pins, and between ground nets and nwell bias pins.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable, when set to true, enables connections between power nets and pwell bias pins, and between ground nets and nwell bias pins.

By default, the variable is set to **false**, and the tool allows only power nets to be connected to power pins and nwell bias pins, and only ground nets to be connected to ground pins and pwell bias pins.

When you set this variable to **true**, in addition to the default PG connections, the tool is also allowed to connect power nets to pwell bias pins, and ground nets to nwell bias pins.

This variable is honored by the commands **derive_pg_connection**, **connect_supply_net**, **check_mv_design**, and **write_verilog**. Commands such as **open_mw_cel**, **save_mw_cel** and **check_database** also consider this variable when they check PG connections in the CEL view.

SEE ALSO

```
connect_supply_net(2)
derive_pg_connection(2)
```

mv_upf_tracking

Controls whether the UPF tracking feature is enabled in the current session.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls whether the UPF tracking feature is enabled in the current session. The default value of this variable is **true**.

When this variable is set to **true** (the default), the user-specified UPF commands are tracked to ensure that their original order is preserved. Also, the UPF' file written by the tool contains two sections. The first section contains the user-specified commands and the second section contains the tool-generated commands. The beginning of the tool-generated section is marked by the following variable setting:

```
set derived_upf true
```

Similarly the end of the tool-generated section is marked by the following variable setting:

```
set derived_upf false
```

To ensure that user-specified UPF content is preserved, this variable has the following usage restrictions:

1. Changing the value of this variable from **true** to **false** is allowed. However, when the variable is changed from **true** to **false**, the variable is marked read-only. This is to ensure that the variable cannot be changed to **true** again.

2. Changing the value of this variable from **false** to **true** is not allowed when the design has UPF constraints.

mv_use_std_cell_for_isolation

Allows the tool to insert standard cells as UPF isolation cells.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable determines whether standard cells can be used as isolation cells. When the variable is set to **false** (the default), standard cells are not used as isolation cells.

When the variable is set to **true**, a latch or a two-input NAND, AND, OR, or NOR gate can be used as an isolation cell if the **ok_for_isolation** lib_cell attribute is set to **true** for the cell, and the **isolation_cell_enable_pin** lib_pin attribute is set to **true** on the enable pin. Based on the isolation sense and clamp value of the isolation strategy, the tool selects an appropriate standard cell for use as an isolation cell and connects it to the isolation supply.

SEE ALSO

```
set_isolation(2)
set_isolation_control(2)
```

mw_allow_rect_and_polygon_in_def

Controls whether the **read_def** and **write_def** commands support rectangles and polygons in the special nets sectioni of the DEF file.

TYPE

Boolean

DEFAULT

true

GROUP

def

DESCRIPTION

Controls whether the **read_def** and **write_def** commands support rectangles and polygons in the special nets section of the DEF file. The polygon edges must be parallel to the x-axis and the y-axis or they must be at a 45-degree angle to the axes.

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

SEE ALSO

```
read_def(2)
write_def(2)
```

mw_allow_unescaped_slash_in_pin_name

Controls whether the **get_pins** command allows unescaped slash characters in the input pin name.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether the **get_pins** command allows unescaped slash characters in the input pin name.

When **false** (the default), the **get_pins** command does not allow any unescaped slash characters in the input pin name and considers the rightmost unescaped slash (/) in the input pattern as the hierarchical delimiter between the cell instance name and the pin name.

When set to **true**, the **get_pins** command allows the input pin name to contain unescaped slash characters and the rightmost unescaped slash in the input name is not necessarily the hierarchical delimiter between the cell instance name and the pin name.

Note:
Enabling the support for allowing unescaped slash characters in the port instance reference names could have the following side effects:

- Significant runtime overhead due to more sophisticated name matching.
- Ambiguity in name matching if there is more than one pin name in the design that matches the input name after the escaping characters in their names are removed.

Therefore, you should enable this capability only if the input pin instance name contains unescaped slash characters.

EXAMPLES

In the following example, the cell instance name is Inst_00 and the pin name is high/test.

```
prompt> set mw_allow_unescaped_slash_in_pin_name false
false
prompt> get_pins {{Inst_00/high\/test}}
{Inst_00/high/test}
prompt> get_pins Inst_00/high/test
Warning: No pin objects matched 'Inst_00/high/test' (SEL-004)

prompt> set mw_allow_unescaped_slash_in_pin_name true
true
prompt> get_pins {{Inst_00/high\/test}}
{Inst_00/high/test}
prompt> get_pins Inst_00/high/test
{Inst_00/high/test}
```

SEE ALSO

get_pins(2)

mw_attr_value_extra_braces

Controls whether extra braces are added to the value returned by the **get_attribute** command.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether extra braces are added to the value returned by the **get_attribute** command.

When **false** (the default), no extra braces are added and the return value can be passed directly to another command.

When set to **true**, extra braces '{' and '}' are added to the number list returned by the **get_attribute** command, which prevents you from passing the value directly to another command. You should set this variable to **true** only for backward-compatibility, such as when you are executing an old script. When you set this variable to **true**, the tool issues an MWUI-041 warning message.

EXAMPLES

The following example shows the different values returned by the **get_attribute** command, depending on the value of this variable.

```
prompt> set mw_attr_value_extra_braces false
false
prompt> get_attribute [get_cells U1] bbox
{30.000 30.000} {40.000 40.000}

prompt> set mw_attr_value_extra_braces true
Warning: Extra braces will be added for number list returned from
get_attribute. (MWUI-041)
true
prompt> get_attribute [get_cells U1] bbox
{{30.000 30.000} {40.000 40.000}}
```

SEE ALSO

get_attribute(2)
MWUI-041(n)

mw_attr_value_no_space

Controls whether the route_type and net_type attribute values returned by the **get_attribute** and **report_attribute** commands contain spaces or underscores.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether the route_type and net_type attribute values returned by the **get_attribute** and **report_attribute** commands contain spaces or underscores.

By default (false), the commands return the backward-compatible attribute values, which contain spaces.

When true, the **get_attribute** and **report_attribute** commands return attribute values with underscores, instead of spaces. These route_type attribute values are accepted by the **create_net_shape** and **create_via** commands.

Note that the **set_attribute** command accepts the attribute values returned by the **get_attribute** and **report_attribute** commands, regardless of the setting of this variable.

EXAMPLES

The following example shows how different attribute values are returned, depending on the value of this variable.

```
prompt> set mw_attr_value_no_space false
false
prompt> get_attribute [get_vias VIA#40704] route_type
P/G Ring
prompt> get_attribute [get_nets n300] net_type
Split Clock

prompt> set mw_attr_value_no_space true
prompt> get_attribute [get_vias VIA#40704] route_type
pg_ring
```

```
prompt> get_attribute [get_nets n300] net_type
split_clock
prompt> set rt [get_attribute [get_net_shapes HWIRE#42240] route_type]
prompt> create_net_shape -route_type $rt -type wire -origin {0 0} \
           -length 10 -net VSS -width 5 -layer METAL4
```

SEE ALSO

create_net_shape(2)
create_via(2)
get_attribute(2)
report_attribute(2)

mw_cell_name

Contains the Milkyway design cell name.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable contains the Milkyway design cell name. By default, the string is empty. If given, **read_mdb** reads in the cell specified by this variable, or **read_mdb** modifies this variable to indicate the cell, view, and version of the MDB cell read in. The **write_mdb** command searches for this variable when writing data to an MDB cell. If this variable is set, it writes to the cell specified by this variable, or it writes to a cell using the current design name.

SEE ALSO

mw_def_fill_map

Defines the mapping between the fill data type names in the DEF file and the datatype numbers used in the FILL view in the Milkyway database.

TYPE

string

DEFAULT

""

GROUP

read_def write_def

DESCRIPTION

This variable defines the mapping between the fill data type names used in the DEF file and the datatype numbers used in the FILL view in the Milkyway database. This mapping affects the reading of DEF data with the **read_def** command and the writing of DEF data with the **write_def** command.

For example, the following command sets FILLWIRE data in the DEF file to datatype 5 in the FILL view in the Milkyway database, and similarly sets the datatype numbers for the DRCFILL and OPC data types.

```
prompt> set_app_var mw_def_fill_map {{FILLWIRE 5} {DRCFILL 8} {OPC 10}}
```

When you use the **read_def** command, FILLWIRE data in the DEF file is mapped to the fill layer with a datatype number of 5 in the Milkyway database. When you use the **write_def** command, any fill data having a datatype value of 5 in the FILL view is written out as FILLWIRE data in the DEF file. IC Validator uses the datatype number to differentiate the types of fill data.

Datatype numbers are integers ranging from 0 to 255. The default datatype number is 0, which is the default datatype of the fill layer. If you do not define the mapping with this variable, when you use the **read_def** command, all fill metal is mapped to the fill layer with a datatype of 0; and when you use the **write_def** command, all fill data is mapped to the

plain fill layer in the DEF file, irrespective of the datatype values used in the FILL view of the Milkyway database.

You can set the mapping for one or more of the FILLWIRE, DRCFILL, and OPC data types in the variable. Any data types not set in the variable are mapped to datatype 0.

When you set this variable, it overwrites any previous settings for the variable.

Net shapes with a **route_type** attribute of "Fill Track" support the FILLWIRE, DRCFILL, and OPC FILL data types. The FILLWIRE data type represents PG metal fill, which is included in SPECIALNETS section of the DEF file. The DRCFILL data type represents notch and gap fill, which is used to correct DRC errors. The OPC data type indicates the FILL shapes required for OPC correction during mask generation.

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

SEE ALSO

```
read_def(2)
write_def(2)
```

mw_design_library

Contains the Milkyway design library.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable contains the Milkyway design libraries. The default is the empty string. When this variable is set, the Milkyway design is stored in the directory specified by the variable. If this variable is set, it will be used by the **read_milkyway**, **write_milkyway** and **create_mw_design** commands.

This variable needs to be set for a smooth data transfer to Milkyway.

SEE ALSO

mw_disable_escape_char

Disables the escape characters for hierarchy delimiters.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls the behavior of escape characters for hierarchies. This variable is used only by the **read_mdb** and **write_mdb** commands.

By default, the tool always escapes the hierarchy delimiters if they are part of names.

When set to **false**, it enables escape characters for hierarchies, similar to setting "No Backslash Insertion to avoid Hier Name Collisions" in Astro.

Make sure that the design does not have name collisions.

SEE ALSO

mw_hdl_bus_dir_for_undef_cell

Specifies how to determine the bus direction for an undefined cell.

TYPE

integer

DEFAULT

0

GROUP

hdl_variables

DESCRIPTION

If any port of any undefined cell is a bus, verilog2cel Verilog reader creates a bus port for the undefined cell and instantiates the cell. This variable specifies the direction of the bus.

For example, assume an undefined cell is instantiated similar to the following:

```
sub si (.port(net[x:y]),...);
```

The number of bus bits = r = |x-y|+1 defines the bus origin according to the value given:

```
0 (From Connection)
x > y type [r:0] port
x < y type [0:r] port
```

1 (Descending)

type [r:0] port

2 (Ascending) type [0:r] port

To determine the current value of this variable, use the **printvar variable_name_filler** command. For a list of all HDL variables and their current values, use **print_variable_group hdl**.

SEE ALSO

`enable_cell_based_verilog_reader(3)`

mw_hdl_expand_cell_with_no_instance

Determines whether or not to expand netlist cells without instances.

TYPE

Boolean

DEFAULT

false

GROUP

hdl_variables

DESCRIPTION

This variable determines whether or not the verilog2cel Verilog reader expands netlist cells that do not contain physical descriptions. By default, no expansion occurs when you have netlist instances with no physical description. Set this variable to **true** if you have netlist instances containing only nets.

To determine the current value of this variable, use the **printvar mw_hdl_expand_cell_with_no_instance** command. For a list of all HDL variables and their current values, use **print_variable_group hdl**.

SEE ALSO

`enable_cell_based_verilog_reader(3)`

mw_hdl_verilog2cel_variables

TCL Variables for the Milkyway cell based verilog reader (verilog2cel). Specify certain variables so that to make verilog2cel behaves as desired.

SYNTAX

```
enable_cell_based_verilog_reader
mw_hdl_allow_dirty_netlist
mw_hdl_create_multi_pg_net
mw_hdl_bus_dir_for_undef_cell
mw_hdl_expand_cell_with_no_instance
mw_hdl_stop_at_FRAM_cells

Boolean enable_cell_based_verilog_reader = "false" (default)
Boolean mw_hdl_allow_dirty_netlist = "false" (default)
Boolean mw_hdl_create_multi_pg_net = "true" (default)
Boolean mw_hdl_bus_dir_for_undef_cell = 0 (default)
Boolean mw_hdl_expand_cell_with_no_instance = "false" (default)
Boolean mw_hdl_stop_at_FRAM_cells = "false" (default)
```

DESCRIPTION

These variables affect the **read_verilog** command when invoking the Milkyway-cell-based Verilog reader to import designs. In addition to these variables, the **mw_current_design** variable is used to indicate the net names for power (1'b1) and ground (1'b0).

enable_cell_based_verilog_reader

Enables the Milkyway-cell-based Verilog reader. See **enable_cell_based_verilog_writer(3)** for details.

mw_hdl_allow_dirty_netlist

Enables the Milkyway-cell-based Verilog reader to handle dirty netlists. See **mw_hdl_allow_dirty_netlist(3)** for details.

mw_hdl_create_multi_pg_net

Generates multiple power and ground nets. See **mw_hdl_create_multi_pg_net(3)** for details.

mw_hdl_bus_dir_for_undef_cell

Specifies the bus direction for undefined cells. See **mw_hdl_bus_dir_for_undef_cell(3)** for details.

mw_hdl_expand_cell_with_no_instance

Enables the Milkyway-cell-based Verilog reader to expand cell hierarchies that have no cell instances. See **mw_hdl_expand_cell_with_no_instance(3)** for details.

mw_hdl_stop_at_FRAM_cells

Tells the Milkyway-cell-based Verilog reader to honor FRAM view cells. See **mw_hdl_stop_at_FRAM_cells(3)** for details.

SEE ALSO

`read_verilog(2)`

mw_reference_library

Contains the Milkyway reference libraries.

TYPE

list

DEFAULT

""

DESCRIPTION

This variable contains the Milkyway reference libraries. The default is the empty string. By setting this variable, the search_path and the physical_library will be enhanced to use the Milkyway libraries.

SEE ALSO

```
create_mw_lib(2)
open_mw_lib(2)
```

mw_site_name_mapping

Specifies pairs of site names that synchronize a floorplan's site name with a physical library's site name.

TYPE

string

DEFAULT

""

GROUP

physopt

DESCRIPTION

This variable specifies pairs of site names that synchronize a floorplan's site name with a physical library's site name. When you use this variable, the floorplan site name is mapped to the physical library site name during library loading. You must set the variable before library loading.

The syntax for the variable is as follows:

```
set mw_site_name_mapping
{ {floorplan_site_name1 mw_reference_lib_site_name}
  {floorplan_site_name2 mw_reference_lib_site_name} }
```

You must use curly braces in the syntax in IC Compiler; therefore, you should also use curly braces in Design Compiler for compatibility between the tools.

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

EXAMPLE

The following example shows three site names in the floorplan file remapped to unit in the Milkyway reference library. You must specify the exact names in the correct case.

```
prompt> set mw_site_mapping { {CORE unit} {core unit} {core2 unit} }
```

SEE ALSO

mw_support_hier_fill_view

Affects the behavior of the **copy_mw_cel**, **remove_mw_cel**, and **save_mw_cel** commands when they operate on a hierarchical FILL view.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable affects the behavior of the **copy_mw_cel**, **remove_mw_cel**, and **save_mw_cel** commands when they operate on a hierarchical FILL view. This type of view created by the **signoff_metal_fill** command.

By default, this variable is set to **true**, which causes the affected commands to consider whether a FILL view is implemented hierarchically and to act accordingly, as described below for each command. If you set this variable to **false**, each command carries out its operation on the specified view without considering whether it is a hierarchical FILL view.

copy_mw_cel Command

The **copy_mw_cel** command copies a specified FILL view hierarchically when you use the **-hierarchy** option in the command OR when the **mw_support_hier_fill_view** variable is set to **true** (even if the **-hierarchy** option is not used). The command copies all lower-level FILL views as well as the top-level FILL view. If copying any of these views would overwrite an identically named view in the destination library, the whole copy operation is canceled and you get an MWUI-1065 error message.

If the variable is set to false AND the **-hierarchy** option is not used, the **copy_mw_cel** command copies only the top-level FILL view, and you get a MWUI-137 warning message.

remove_mw_cel Command

The **remove_mw_cel** command removes a specified FILL view hierarchically when you use the **-hierarchy** option in the command OR when the **mw_support_hier_fill_view** variable is set to **true** (even if the **-hierarchy** option is not used). The command removes all lower-level FILL views as well as the top-level FILL view.

If the variable is set to false AND the **-hierarchy** option is not used, the **remove_mw_cel** command removes only the top-level FILL view, and you get a MWUI-137 warning message.

save_mw_cel Command

The **save_mw_cel** command saves a specified FILL view hierarchically when you use the **-hierarchy** option in the command OR when the **mw_support_hier_fill_view** variable is set to **true** (even if the **-hierarchy** option is not used). The command saves any open and modified lower-level FILL views as well as the top-level FILL view.

If the variable is set to false AND the **-hierarchy** option is not used, the **save_mw_cel** command saves only the top-level FILL view, and you get a MWUI-137 warning message.

SEE ALSO

```
copy_mw_cel(2)
remove_mw_cel(2)
save_mw_cel(2)
signoff_metal_fill(2)
MWUI-137(n)
MWUI-1065(n)
```

mw_use_allowable_orientation

Specifies the method used by the tool to determine the allowable orientations of cells during placement legalization.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable selects the method used by the IC Compiler tool to determine the allowable orientations of cells being optimized for placement by the **legalize_placement** command.

Leave this variable set to **false** (the default) to allow the LEF-defined cell symmetry to determine the allowable cell orientations during placement legalization. Set this variable to **true** to use the allowed orientations previously specified for cells in the Milkyway Environment tool.

If this variable is set to **false**, the tool determines the allowed orientations by considering the symmetry of the cell as defined by the SYMMETRY statement in the LEF language, if the cell data was derived from physical data in LEF format. In the absence any SYMMETRY definition, the tool assumes "X Y" symmetry for a standard cell or "X Y R90" symmetry for a macro cell. "X" symmetry means that the cell can be flipped about the x-axis (and similarly for "Y" symmetry). "R90" symmetry means that the cell can be rotated 90 degrees.

If this variable is set to **true**, the tool reads in and uses the allowable orientation information previously defined for the cell in the Milkyway Environment tool, using the **dbSetStdCellOrient** command for a standard cell or the **dbSetMacroCellOrient** command for a macro. For information about using these commands, see the online help in the Milkyway Environment Tool. The allowable orientation information is stored in the FRAM view of the cell. To use this feature, you must set the variable to **true** in the IC Compiler tool before you open the design.

EXAMPLES

The following example shows how to set the variable.

```
prompt> set_app_var mw_use_allowable_orientation true
```

SEE ALSO

```
check_legality(2)
legalize_placement(2)
```

mwdc_allow_higher_mem_usage

Specifies the interval at which the tool flushes in-memory data to disk during optimization, allowing you to control the tradeoff between memory usage and runtime.

TYPE

integer

DEFAULT

0

DESCRIPTION

This variable specifies the interval at which the tool flushes in-memory data to disk during optimization, allowing you to control the tradeoff between memory usage and runtime. It affects the flushing of in-memory data during execution of the commands **place_opt**, **clock_opt**, **route_opt**, **create_placement**, **legalize_placement**, **compile_clock_tree**, and similar commands that perform optimization.

You can set this variable to 0, 1, or 2:

- The value **0** (default) causes the tool to flush in-memory data to disk as often as possible during optimization, resulting in the smallest memory usage, at the cost of greater runtime.
- The value **1** causes to tool not flush in-memory data to disk at all during optimization, resulting in the least runtime, at the cost of higher memory usage. Flushing of in-memory data to disk occurs only when the **save_mw_cel** command is executed. Note that the advantage of runtime reduction is lost if the working machine runs out of physical memory and must use swap disk space as virtual memory; in that case, the runtime could be even worse than using mode 0.
- The value **2** causes the tool to flush in-memory data to disk adaptively in a balanced manner, resulting in less runtime than mode 1 and less memory usage than mode 2. Flushing of data occurs only when memory usage approaches the maximum available physical memory or when the accumulated changed data reaches a certain threshold.

SEE ALSO

```
clock_opt(2)
place_opt(2)
psynopt(2)
route_opt(2)
```

net_attributes

Contains attributes that can be placed on a net.

DESCRIPTION

Contains attributes that can be placed on a net.

To set an attribute, use the command identified in the individual description of that attribute. If an attribute is "read-only," the user cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Net Attributes

ba_net_resistance

A floating point number that specifies the back-annotated net resistance on a net. Set with **set_resistance**.

bus_name

If the net is bussed, returns the full hierarchical name of the net bus. This attribute is **read only** and cannot be set by the user.

dont_touch

Identifies nets to be excluded from optimization. Values are *true* (the default) or *false*. Nets with the **dont_touch** attribute set to *true* are not modified or replaced during **compile**. Set with **set_dont_touch**.

groute_length

Get the global routing length of the net if global route is done for the net.

is_bussed

Returns true if the net is part of a bussed net. This attribute is **read only** and cannot be set by the user.

is_test_circuitry

Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry. This attribute is read-only and cannot be set by the user.

load *

A floating point number that specifies the wire load value on a net. The total load on a net is the sum of all the loads on pins, ports, and wires associated with that net. This attribute represents only the wire load; pin and port loads are not included in the attribute value unless the attribute *subtract_pin_load* is set to **true** for the same net. Set with **set_load**.

route_mode

The rerouting mode of a net. The attribute could be **normal**, **minorchange**, or**freeze**. The attribute could be set by **set_net_routing_rule** command with **-reroute** option.

static_probability

A floating point number that specifies the percentage of time that the signal is in the logic 1 state; this information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

subtract_pin_load

Causes **compile** to reduce the wire load value of a net by an amount equal to its pin load. Specifies that the **load** attribute includes the capacitances of all pins on the net. If the resulting wire load is negative, it is set to zero. Set with **set_load -subtract_pin_load**.

route_guide_group_id

Specifies the route guide group id for a flat net. The data type of **route_guide_group_id** is int. The valid value range of **route_guide_group_id** is 1~8. This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

switching_activity

A positive floating point number that specifies the switching activity; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**; if this attribute is not set, **report_power** will use the default value of 2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

wired_and

One of a set of two wired logic attributes that includes **wired_or**. When present and set to *true*, **wired_and** determines that the associated net has more than one driver and implements a wired AND function. Wired logic attributes cannot be manually set by the user. To cause wired logic attributes to be added to a netlist design that contains multiply-driven nets, you have two alternatives: 1. execute **compile** or **translate** on the design; or 2. specify the wired logic types using a resolution function in the HDL file.

wired_or

One of a set of two wired logic attributes that includes **wired_and**. When present and set to *true*, **wired_or** determines that the associated net has more than one driver and implements a wired OR function. Wired logic attributes cannot be manually set by the user. To cause wired logic attributes to be added to a netlist design that contains multiply-driven nets, you have two alternatives: 1. execute **compile** or **translate** on the design; or 2. specify the wired logic types using a resolution function in the HDL file.

within_block_abstraction

Specifies whether the net is part of the block abstraction.

This attribute is read-only and cannot be modified.

within_ilm

Specifies whether the net is part of an ILM.

This attribute is read-only and cannot be modified.

is_bus

Tells if the net is a bus net.

This attribute is read-only and cannot be modified.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
attributes(3)
```

physical_bus_attributes

Contains attributes related to physical buses.

DESCRIPTION

Contains attributes related to physical buses.

You can use the **get_attribute** command to determine the value of an attribute and use the **report_attribute** command to get a report of all attributes on a specified physical bus. To list the definitions of the attributes, use the **list_attribute -class physical_bus -application** command.

Physical Bus Attributes

cell_id

Specifies the Milkyway design ID in which a physical bus object is located.

This attribute is read-only.

name

Specifies the name of a physical bus object.

This attribute is read-only.

num_nets

Specifies the number of nets in a physical bus object.

You can get the nets associated with a physical bus object by using the **get_nets -of_object <physical_bus_object>** command.

This attribute is read-only.

object_class

Specifies the object class name of a physical bus object, which is **physical_bus**.

This attribute is read-only.

object_id

Specifies the object ID in the Milkyway design file.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
```

physical_lib_pin_attributes

Describes the attributes related to physical library pins.

DESCRIPTION

This man page describes the attributes related to physical library pins.

You can use the **get_attribute** command to determine value of an attribute, and use the **report_attribute** command to get a report of all attributes on a specified object. To see the attributes defined for physical library pins, use the **list_attribute -class physical_lib_pin -application** command.

All of these attributes are read-only.

Physical Library Pin Attributes

cell_id

Specifies Milkyway design ID in which a physical_lib_pin object is located.

The data type of the **cell_id** attribute is integer.

cell_name

Specifies name of the Milkyway design in which a physical_lib_pin object is located.

The data type of the **cell_name** attribute is string.

direction

Specifies the direction of a physical_lib_pin object.

The data type of the **direction** attribute is string.

full_name

Specifies the full name of a physical_lib_pin object. The full name of a physical_lib_pin object is composed of the library name (**lib_name** attribute), the cell name (**cell_name** attribute), and the pin name (**name** attribute).

The data type of the **full_name** attribute is string.

is_diode

Indicates whether the specified object is a diode.

The data type of the **is_diode** attribute is Boolean.

lib_name

Specifies the library name of a physical_lib_pin object.

The data type of the **lib_name** attribute is string.

lib_path

Specifies the UNIX path name of the library that contains the physical_lib_pin object.

The data type of the **lib_path** attribute is string.

name

Specifies the name of a physical_lib_pin object.

The data type of the **name** attribute is string.

object_class

Specifies the object class name of a physical_lib_pin, which is **physical_lib_pin**.

The data type of the **object_class** attribute is string.

object_id

Specifies the ID of the physical_lib_pin object in the Milkyway design database.

The data type of the **object_id** attribute is integer.

pin_type

Specifies the pin type of a physical_lib_pin object.

The data type of the **pin_type** attribute is string.

pg_pin_weight

A float attribute representing the PG pin weight. It should be between 0.1 and 25.0. And the fraction precision is 0.1, e.g. 0.11 will be cut off as 0.1.

SEE ALSO

```
get_attribute(2)
list_attribute(2
report_attribute(2)
```

physopt_area_critical_range

Specifies a margin of slack for cells during area optimization. If a cell has a slack less than the area critical range, area optimization is not done for the cell.

TYPE

float

DEFAULT

-1.04858e+06

GROUP

physopt

DESCRIPTION

The **physopt_area_critical_range** variable specifies a margin of slack for cells during area optimization. If a cell has a slack less than the area critical range, area optimization is not done for the cell. The default value is minus infinity; that is, all cells are optimized for area during area recovery.

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

SEE ALSO

`physopt_ultra_high_area_effort(3)`

physopt_check_site_array_overlap

Checks for multi-type site array overlapping in floorplan, within a certain threshold.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

DESCRIPTION

If multiple types of site arrays are defined in the floorplan, the detail placer requires that the overlapping occur within a certain threshold, to ensure best quality of results. You set the threshold with the **physopt_row_overlap_threshold** variable.

The default value of **physopt_check_site_array_overlap** is *true*. Setting the value of this variable to *false* disables checking.

SEE ALSO

`physopt_row_overlap_threshold(3)`

physopt_cpu_limit

This variable is obsolete. Use **set_physopt_cpulimit_options** command instead.

TYPE

float

DEFAULT

0

GROUP

physopt

DESCRIPTION

Remove this obsolete variable and replace it with the new command, **set_physopt_cpulimit_options**. For details, see the **set_physopt_cpulimit_options** man page.

SEE ALSO

`set_physopt_cpulimit_options(2)`

physopt_create_missing_physical_libcells

Directs the tool to create dummy physical descriptions of missing physical library cells.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

This variable enables the tool to create a dummy physical cell if a cell is missing from the user-supplied physical libraries.

The tool uses a simple physical description for these dummy physical cells. This is mainly required in the exploration flow so that the tool does not stop when you do not have the final physical libraries. This is not intended as a replacement for specifying the correct physical libraries. For a final and accurate solution, you must specify the correct physical libraries.

The dummy physical cells created by the tool are not persistent. They exist only in the current session and are not stored in the database. These cells are recreated in a new session, if they are still required.

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

physopt_delete_unloaded_cells

Controls whether the **physopt** command deletes unloaded cells, including both sequential and combinational cells.

TYPE

Boolean

DEFAULT

true

GROUP

Physopt

DESCRIPTION

Controls whether the **physopt** command deletes unloaded cells. A design can contain cells that drive no loads. During **physopt**, the logic driven by a cell might be optimized away, resulting in an inferred no-load, or no path to any primary output. By default, **physopt** deletes such cells. To retain these cells, set this variable, **physopt_delete_unloaded_cells**, to *false*.

SEE ALSO

physopt_enable_power_optimization

Enables power optimization.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

DESCRIPTION

The **physopt_enable_power_optimization** variable enables and disables power optimization.

To disable power optimization, set the value to false.

To enable power optimization, set the value to true (the default).

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

prompt> **printvar physopt_enable_power_optimization**

SEE ALSO

physopt_enable_via_res_support

Enables and disables the support of via resistance for virtual route RC estimation.

TYPE

Boolean

DEFAULT

true

GROUP

physopt

DESCRIPTION

This variable enables the support of via resistance for virtual route RC estimation when set to **true**.

If you want to disable the support of via resistance for virtual route RC estimation, set the value of this variable to **false** (the default value).

To see the current value of this variable, use the **printvar physopt_enable_via_res_support** command.

physopt_hard_keepout_distance

Specifies the keepout distance used by the **physopt**, **create_placement**, and **legalize_placement** commands.

TYPE

float

DEFAULT

0

GROUP

physopt

DESCRIPTION

This variable specifies the keepout distance used by the **physopt**, **create_placement**, and **legalize_placement** commands. Specify the distance in micron units.

To prevent congestion, it is useful to mark an area in the design that surrounds a fixed macro as a hard keepout area. If you specify an area as a hard keepout area, the tool does not place any cells in that area. Define the area to be marked by setting a value for this variable, which the tool then uses to inflate the fixed cell's rectangle the same distance in all four directions.

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

SEE ALSO

```
create_placement(2)
legalize_placement(2)
```

physopt_heterogeneous_site_array

Enables the detailed placer to handle floorplan with multi-height site arrays in multi-voltage mode.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

If the arrays of multiple types of sites with different site-heights are defined in the floorplan, to enable the detailed placer to place cells properly on those sites this variable needs to be set to true.

If it is set to false, detailed placer ignores the multi-height site types even if they are present in the floorplan.

physopt_ignore_lpin_fanout

Ignores max fanout constraints that are specified in the library.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

This variable instructs tools to ignore max_fanout violations from the constraints in the library. Only optimization ignores those constraints and the **report_constraint** command still reports the violation.

SEE ALSO

`report_constraint(2)`

physopt_ignore_structure

Controls whether the tool ignores the relative placement groups.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

This variable controls whether the tool ignores the relative placement groups.

When **false** (the default), the tool carries out its default behavior and utilizes the relative placement groups.

When set to **true**, the tool does not perform relative placement on the design.

To see the current value of this variable, enter

```
prompt> printvar physopt_ignore_structure
```

SEE ALSO

`place_opt(2)`

physopt_macro_cell_height_threshold

Specifies the threshold value for the tool's detail placer to decide whether an extremely tall standard cell is to be treated as a "hard macro." The tool's detail placer is designed to place standard cells. If the design has some extremely tall standard cells, then the detail placer might not work as well as it does when placing "normal" standard cells. Thus, by default, if an extremely tall standard cell has a height that is greater than this threshold value, the detail placer treats the standard cell as a "hard macro."

TYPE

integer

DEFAULT

6

GROUP

physopt

DESCRIPTION

This variable specifies the threshold value for the tool's detail placer. This threshold determines whether a standard cell becomes a "hard macro." For example, if a standard cell has a 10x site height, then it is treated as a hard macro. The default threshold value is 6 times the site height. You can set this variable to adjust the "threshold value." The tool's detail placer treats some very tall standard cells as "hard macros" by default.

This variable is allowed only in the this tool.

SEE ALSO

```
check_legality(2)
legalize_placement(2)
psynopt(2)
place_opt(2)
```

physopt_mw_checkpoint_filename

Specifies the name of the file in which to write the Milkyway database containing all the hierarchy of the checkpointed design.

TYPE

string

DEFAULT

""

GROUP

physopt

DESCRIPTION

This variable specifies the name of the file in which to write the Milkyway database containing all the hierarchy of the checkpointed design. The filename must be purely relative. The directory path must be specified in the variable **mw_design_library**.

SEE ALSO

```
psynopt(2)
mw_design_library(3)
```

physopt_new_fix_constants

Determines whether the tool is to observe the maximum capacitance constraint during tie-off optimization.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Determines whether the tool is to observe the maximum capacitance constraint (set with the **max_capacitance** attribute of the **set_max_capacitance** command) during tie-off optimization.

During postprocessing after optimization, the tool maps all generic constant logic cells to tie-off cells. If you set this variable as **true** (the default) during this phase, the tool optimizes the number of tie-off cells required to drive the loads to constant logic value. To derive the number of tie-off cells required, and their locations, the tool considers the max_fanout design rule constraint and the max_cap design rule constraint on the design.

If you set this variable as **false**, the tool considers only the max_fanout design rule constraint (but not the max_cap design rule constraint) in performing the tie-off optimization.

To see the current value of this variable, type

psyn_shell-xg-t> **printvar physopt_new_fix_constants**

SEE ALSO

`set_max_capacitance(2)`

physopt_pin_based_pad

Specifies whether is_pad attribute is set to pin.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

By default, physopt places implicit dont_touch on nets connected to cells which have "is_pad" attribute. If this variable is set to true, then physopt places implicit dont_touch on nets connected to pin which has "is_pad" attribute.

SEE ALSO

physopt_power_critical_range

Specifies a margin of slack for cells during leakage power optimization. If a cell has a slack less than the power critical range, power optimization will not be done for the cell.

TYPE

float

DEFAULT

-1.04858e+06

GROUP

physopt

DESCRIPTION

This variable specifies a power critical range, which is a margin of slack for cells during leakage power optimization. If a cell has a slack less than the power critical range, leakage power optimization is not performed for the cell.

By default, the value of minus infinity indicates that all cells are optimized for leakage power during the leakage power optimization phase of the **physopt** command.

SEE ALSO

`set_max_leakage_power(2)`

physopt_ref_pdef_loaded

Stores the name of the reference loaded during report_cell_displacement -load_pdef. Otherwise, stores the null string.

TYPE

String

DEFAULT

""

GROUP

physopt

DESCRIPTION

This variable stores the name of the reference that PDEF loaded during report_cell_displacement -load_pdef xx.pdef. This is used to obtain the value of the PDEF file inside psyn_shell, and is used to assign values to other temporary variables. The variable gets updated after running report_cell_displacement. However to clear its value, you must run report_cell_displacement -remove_pdef.

SEE ALSO

physopt_row_overlap_threshold

Specifies a threshold for checking multi-type site array overlapping floorplan.

TYPE

float

DEFAULT

0.1

GROUP

physopt

DESCRIPTION

A specified threshold is used to check multi-type site array overlapping in a floorplan. A threshold set at 0.1 means that a 10 percent difference of normal core area is allowed.

It is not recommended that you set the threshold to larger number than 0.5, which means 50 percent difference. A larger setting will adversely affect quality of result.

SEE ALSO

`physopt_check_site_array_overlap(3)`

physopt_tie_const_cells

Connects all unconnected pins in the same hierarchy to one logic cell.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

By default, when the design has some unconnected pins, the tool connects each unconnected pin to a unique logic cell. When this variable is true, the tool connects all unconnected pins in the same hierarchy to one logic cell.

SEE ALSO

physopt_ultra_high_area_effort

Enables very high effort area optimization for potentially better area recovery but with more runtime.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

This variable enables very high effort area optimization during high effort area recovery for potentially better QOR, but with more runtime.

This variable can be used only with -area_effort high. This variable is valid only in Physical Compiler.

SEE ALSO

`physopt_area_critical_range(3)`

pin_attributes

Contains attributes placed on pins.

DESCRIPTION

Contains attributes that can be placed on a pin.

There are a number of commands used to set attributes; however, most attributes can be set with the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute. Otherwise, use **set_attribute**. If an attribute is *read only*, it cannot be set by the user.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Pin Attributes

bus_name

If the pin is bussed, returns the full hierarchical name of the pin bus. This attribute is **read only** and cannot be set by the user.

disable_timing

Disables timing arcs. This has the same affect on timing as not having the arc in the library. Set with **set_disable_timing**.

is_bussed

Returns true if the pin is part of a bussed pin. This attribute is **read only** and cannot be set by the user.

is_diode

Tells if the pin is a diode pin.

max_fall_delay

A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_rise_delay

A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

min_fall_delay

A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay

A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

observe_pin

Specifies the (internal) observe pin name of an LSI Logic scan macrocell (LSI CTV only). This attribute is used by the **write_test** command. Set with **set_attribute**.

pin_direction

Returns the direction of a pin. The value can be **in**, **out**, **inout**, or **unknown**. This attribute cannot be set by the user. You can also use the **direction** attribute to get the direction of the pin.

pin_properties

Lists valid EDIF property values to be attached to different versions of the output pin. The EDIF property values correspond to different output emitter-follower resistance values on the output pin. For details about the use of this attribute, refer to the *Library Compiler Reference Manual*. Set with **set_attribute**.

set_pin

Specifies the (internal) set pin name of an LSI Logic scan macrocell (LSI CTV only). This attribute is used by the **write_test** command. Set with **set_attribute**.

signal_type

Used to indicate that a pin or port is of a special type, such as a **clocked_on_also** port in a master/slave clocking scheme, or a **test_scan_in** pin for scan-test circuitry. Set with **set_signal_type**.

static_probability

A floating point number that specifies the percentage of time that the signal is in the logic 1 state; this information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

test_assume

A string that represents a constant logic value to be assumed for specified pins throughout test design rule checking, test pattern generation, and fault simulation by **check_test**, **create_test_patterns**, and **fault_simulate**. "1", "one", or "ONE" specifies a constant value of logic one; "0", "zero", or "ZERO" specifies a constant value of logic zero. Use **report_test -assertions** for a report on objects that have the **test_assume** attribute set. Set with **set_test_assume**.

test_dont_fault

Specifies pins not faulted during test pattern generation. If no command options are specified, this attribute is set for both stuck-at-0 and stuck-at-1 faults. Set with **set_test_dont_fault**.

test_initial

A string that represents an initial logic value to be assumed for specified pins at the start of test design rule checking and fault simulation by **check_test** and **fault_simulate**. "1", "one", or "ONE" specifies an initial value of logic one; "0", "zero", or "ZERO" specifies an initial value of logic zero. Use **report_test -assertions** for a report on objects that have the **test_initial** attribute set. Set with **set_test_initial**.

test_isolate

Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.

Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

test_require

Specifies a constant, fixed logic value that a pin is required to have during scan test vector generation. The pin maintains the same value for each test vector generated. Use **report_test -assertions** for a report on objects that have the **test_require** attribute set. Set with **set_test_require**.

test_routing_position

Specifies the preferred routing order of the scan-test signals of the identified cells. Set with **set_test_routing_order**.

switching_activity

A positive floating point number that specifies the switching activity; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**; if this attribute is not set, **report_power** will use the default value of

2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

true_delay_case_analysis

Specifies a value to set all or part of an input vector for **report_timing -true** and **report_timing -justify**. Allowed values are *0*, *1*, *r* (rise, X to 1), and *f* (fall, X to 0). Set with **set_true_delay_case_analysis**.

within_block_abstraction

Specifies whether the pin is part of the block abstraction.

This attribute is read-only and cannot be modified.

within_ilm

Specifies whether the pin is part of an ILM.

This attribute is read-only and cannot be modified.

is_bus

Tells if the pin is a bus pin.

This attribute is read-only and cannot be modified.

clock_latency_fall_max

A float attribute representing the user-specified clock network latency for max fall. This attribute is only defined on those pins where **set_clock_latency** has been directly set.

clock_latency_fall_min

A float attribute representing the user-specified clock network latency for min fall. This attribute is only defined on those pins where **set_clock_latency** has been directly set.

clock_latency_rise_max

A float attribute representing the user-specified clock network latency for max rise. This attribute is only defined on those pins where **set_clock_latency** has been directly set.

clock_latency_rise_min

A float attribute representing the user-specified clock network latency for min rise. This attribute is only defined on those pins where **set_clock_latency** has been directly set.

hold_uncertainty

Specifies a negative uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this pin. Set with **set_clock_uncertainty -hold**.

setup_uncertainty

Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive anout of this pin. Set with **set_clock_uncertainty -setup**.

pg_pin_weight

A float attribute representing the PG pin weight. It should be between 0.1 and 25.0. And the fraction precision is 0.1, e.g. 0.11 will be cut off as 0.1.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)
```

pin_shape_attributes

Contains attributes related to pin shape.

DESCRIPTION

Contains attributes related to pin shape.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified physical bus. Specified with **list_attribute -class pin_shape -application**, the definition of attributes can be listed.

Pin Shape Attributes

access_direction

Specifies the accessible direction(s) of a pin shape.

This attribute is read-only.

The valid values can be:

- **Left** indicates the object can be accessed from left.
- **Right** indicates the object can be accessed from right.
- **Up** indicates the object can be accessed from above.
- **Down** indicates the object can be accessed from below.
- **Left Right** indicates the object can be accessed from left and right.
- **Left Up** indicates the object can be accessed from left and above.
- **Left Down** indicates the object can be accessed from left and below.
- **Right Up** indicates the object can be accessed from right and above.

- **Right Down** indicates the object can be accessed from right and below.
- **Up Down** indicates the object can be accessed from above and below.
- **Left Right Up** indicates the object can not be access from below.
- **Left Right Down** indicates the object can not be access from above.
- **Left Up Down** indicates the object can not be access from right.
- **Right Up Down** indicates the object can not be access from left.
- **Left Right Up Down** indicates the object can be accessed from all directions.

Note that this attribute is undefined on a diode object. You will get warning message with message ID "ATTR-3" when you try to get its value on diode pin shape.

base_name

Specifies the base name of a pin shape. The **base_name** of a pin shape is similar with the **name** of the corresponding terminal in the child MW design.

This attribute is read-only.

bbox

Specifies the bounding-box of a pin shape. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The *bbox* of a pin shape is calculated by the **origin** and **orientation** of its cell and the actual **bbox** of its corresponding terminal from the child MW design.

This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding-box of a pin shape.

The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ll** of a pin shape by accessing the first element of its **bbox**.

This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a pin shape.

The data type of **bbox_llx** is double.

This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a pin shape.

The data type of **bbox_lly** is double.

This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a pin shape.

The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ur** of a pin shape by accessing the second element of its **bbox**.

This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a pin shape.

The data type of **bbox_urx** is double.

This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a pin shape.

The data type of **bbox_ury** is double.

This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a pin shape object is located.

This attribute is read-only.

direction

Specifies the direction of a pin shape.

The valid values can be: in, out, inout and tristate.

This attribute is read-only.

double_pattern_mask_constraint

Specifies the coloring information of the pin_shape. This information is needed in the double patterning flow.

The valid values can be: any_mask, mask1_soft, mask1_hard, mask2_soft, mask2_hard and same_mask.

This attribute is read-only.

full_name

Specifies the full name of a pin shape.

The full name of a pin shape consists of the full name of its owned cell and the name of the corresponding terminal from child MW design.

This attribute is read-only.

is_diode

Indicates whether a pin shape is diode.

The data type of **is_diode** is boolean.

This attribute is read-only.

is_fixed

Indicates whether a pin shape is fixed or not.

This attribute is read-only.

layer

Specifies the layer name of a pin shape.

This attribute is read-only.

multiple_pattern_mask_constraint

Specifies the coloring information of the pin_shape. This information is needed in the multiple patterning flow.

The valid values can be: any_mask, mask1_soft, mask1_hard, mask2_soft, mask2_hard, mask3_soft, mask3_hard and same_mask.

This attribute is read-only.

name

Specifies name of a pin shape. It's identical as the attribute **full_name**.

This attribute is read-only.

number_of_points

Specifies the number of points to illustrate the boundary of a pin shape.

You can refer to the attribute **points**. The list length of **points** is the value of **number_of_points**.

This attribute is read-only.

object_class

Specifies object class name of a pin shape, which is **pin_shape**.

This attribute is read-only.

points

Specifies points of the boundary of a pin shape. A pin shape can be a rectangle, a rectilinear polygon, or multiple rectangles.

When a pin shape is either a rectangle or a rectilinear polygon, its **points** is represented by a list of points. The last element of the list is the same as the first element.

When a pin shape consists of multiple rectangles, its **points** is represented by a list of points of rectangles. Every five points represent one rectangle.

The *points* of a pin shape is calculated by the **origin** and **orientation** of its cell and the actual **points** of its corresponding terminal from the child MW design.

This attribute is read-only.

status

Specifies the PR status of a pin shape.

The valid values are: fixed, placed, cover, and unplaced.

This attribute is read-only.

Note that this attribute is undefined on a diode object. You will get warning message with message ID "ATTR-3" when you try to get its value on diode pin shape.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
```

placement_blockage_attributes

Contains attributes related to placement blockage.

DESCRIPTION

Contains attributes related to placement blockage.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class placement_blockage -application**, the definition of attributes can be listed.

Placement Blockage Attributes

affects

Specifies the affects of a placement blockage, which is placement.

The data type of **affects** is string.

This attribute is read-only.

area

Specifies area of a placement blockage.

The data type of **area** is float.

This attribute is read-only.

bbox

Specifies the bounding-box of a placement blockage. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

blocked_percentage

Specifies the percentage blockage for a partial blockage.

The data type of **blocked_percentage** is integer.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

cell_id

Specifies Milkyway design ID in which a placement blockage object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

layer

Specifies layer name of a placement blockage.

The data type of **layer** is string.

Its valid values are:

- **PlaceBlockage**
- **SoftPlaceBlk**
- **pinBlockage**
- **MacroBlockage**
- **PartialPlaceBlk**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name

Specifies name of a placement blockage object.

The data type of **name** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

object_class

Specifies object class name of a placement blockage, which is **placement_blockage**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

The data type of **object_id** is integer.

This attribute is read-only.

pin_blockage_layers

Specifies the layers for which routing to pins are blocked. This attribute only applies on pin blockage type.

The data type of **pin_blockage_layers** is string.

This attribute is read-only.

type

Specifies type of a placement blockage.

The data type of **type** is string.

Its valid values are:

hard

soft

pin

hard_macro

partial

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

is_reserved_placement_area

The attribute is true if the placement blockage has the property "is_reserved_placement_area".

The attribute is read-only.

no_register is a writable boolean attribute. If true, it specifies that register cells are prohibited within the blockage. All other cells are allowed.

buffer_only is a writable boolean attribute. If true, it specifies that non-buffer cells are prohibited within the blockage. Buffers and inverters are allowed.

no_rp_group is a writable boolean attribute. If true, it specifies that rp groups are prohibited within the blockage. All other cells are allowed.

category_name is a writable string attribute. If set, it specifies the name of a boolean user-defined attribute. Use this attribute to specify which cells or references are prohibited within the blockage. See **create_placement_blockage** for details.

no_register, **buffer_only**, **no_rp_group** and **category_name** are mutually exclusive, i.e. they may not be used together on the same placement blockage. They may only be used on partial type blockages. See **create_placement_blockage** for more information.

buffer_only and **category_name** require the **redefined blockage behavior** feature, enabled as follows:

set placer_enable_redefined_blockage_behavior true

SEE ALSO

```
create_placement_blockage(2)
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
placer_enable_redefined_blockage_behavior(3)
```

placer_channel_detect_mode

Controls the treatment of channel areas during coarse placement.

TYPE

string

DEFAULT

false

DESCRIPTION

This variable controls the treatment of channel areas during coarse placement.

Valid values are **auto**, **true**, or **false**.

By default (**false**), the tool does not treat channel areas differently.

When you set this variable to **true**, the tool analyzes the area available for placement and classifies narrow parts as channels. The tool reduces the maximum cell density allowed in channel areas during coarse placement to help reduce congestion.

When you set this variable to **auto**, the tool considers the amount of channel area in the design. If the tool determines that the design has little or no channel area, it does not treat channel areas differently. If the tool determines that the design has a substantial amount of channel area, it reduces the maximum cell density allowed in channel areas during coarse placement to help reduce congestion.

Some increase in runtime and memory usage is expected when this feature is active. Setting this variable to **auto** only incurs the runtime and memory penalties when the design has a substantial amount of channel area.

SEE ALSO

```
place_opt(2)
create_placement(2)
```

placer_congestion_effort

Controls which congestion estimator is used during congestion-driven placement in the **place_opt -congestion** or **compile_ultra -spg** flow.

TYPE

string

DEFAULT

auto

DESCRIPTION

The **placer_congestion_effort** variable controls which congestion estimator is used during congestion-driven placement in the **place_opt -congestion** or **compile_ultra -spg** flow.

Valid values are **auto**, **none**, **low**, **medium**, or **high**. By default (**auto**), the tool first runs a fast internal estimator to determine how much congestion there is in the placement. If the tool determines that there is not much congestion, it uses the fast estimator's map to drive congestion-driven placement. If the tool determines that there is much congestion, it calls the Zroute global router to do a more accurate congestion estimation and uses that map to drive congestion-driven placement.

When you set this variable to **none** or **low**, the tool always uses the internal estimator for congestion estimation.

When you set the variable to **medium**, the tool always uses the Zroute global router for congestion-driven placement.

When you set this variable to **high**, the tool always uses the Zroute global router for congestion-driven placement and works harder on congestion removal.

SEE ALSO

```
place_opt(2)
placer_enable_enhanced_router(3)
```

placer_disable_auto_bound_for_gated_clock

Determines whether automatic group bounding is disabled for gated clocks created by Power Compiler.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

The gated-clock method reduces a design's power consumption by controlling the switching activity from the clock signal to flip-flop registers. The automatic bound function that this variable enables (when set to *false*, the default is true) works only for the gated-clock elements Power Compiler creates. The **placer_disable_auto_bound_for_gated_clock** variable does not affect your design unless you use the clock-gating functionality of Power Compiler.

To reduce clock skew, place the clock-gating element and the flip-flops it controls close together in your design. The tool automatically creates a group bound for each clock-gating element and the flip-flops the element drives. This group bound helps place the flip-flops closer to the clock-gating element.

If this variable remains true, the tool does not create a group bound for the clock-gating elements created with Power Compiler.

SEE ALSO

```
remove_bounds(2)
report_bounds(2)
create_bounds(2)
```

placer_disable_macro_placement_timeout

Prevents the coarse placer from timing out during macro placement.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Use this variable to prevent the coarse placer from timing out during macro placement when running **create_placement** with floating macros.

By default, macro placement exits early if runtime is too long when compared to the rest of coarse placement. This is to prevent excessively long runtimes that can occur when there are too many floating macros, too many constraints on macros, or bad constraints. The result is that some macros may be placed illegally.

When this variable is set to true, the placer will let macro placement run to completion regardless of runtime.

The default value of this variable is false.

SEE ALSO

```
set_mpc_macro_array(2)
create_placement(2)
```

placer_dont_error_out_on_conflicting_bounds

Prevents the coarse placer from exiting with an error when it detects conflicting bounds.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Use this variable to prevent the tool from exiting with an error when it detects conflicting bounds when running **place_opt** or **create_placement**.

By default, the placer will automatically adjust the conflicting groupbounds and continue instead of exiting with an error.

When this variable is set to false, the placer checks for conflicting movebounds and groupbounds and errors out if a conflict is detected. These bounds can be user-specified bounds or bounds generated by the tool.

The default value of this variable is true.

SEE ALSO

```
create_bounds(2)
create_placement(2)
place_opt(2)
```

placer_enable_enhanced_router

Enables Zroute-based congestion-driven placement to perform more accurate, congestion-aware placement.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When the **placer_enable_enhanced_router** variable is set to **true**, the tool uses Zroute-based congestion-driven placement instead of congestion estimation based on virtual routing, which estimates congestion based on wire-routing predictions. Zroute-based congestion estimation is enabled only on designs that meet the internal congestion threshold. Any designs that are not congested enough to meet the threshold are estimated by virtual routing (the default).

Design Compiler Graphical and IC Compiler use the same internal congestion thresholds. As a result, Zroute-based congestion-driven placement provides better optimization and congestion correlation between Design Compiler Graphical and IC Compiler than virtual routing.

Note that the **set_congestion_options -layer** command is not supported when **placer_enable_enhanced_router** is enabled.

Use the **create_route_guide** command to pass layer availability constraints to the global router.

You can use this feature in conjunction with any command that invokes congestion-driven placement, such as **place_opt -congestion**, **psynopt -congestion**, and others.

Use the following command to determine the current value of the variable:

```
prompt> printvar placer_enable_enhanced_router
```

SEE ALSO

```
place_opt(2)
psynopt(2)
set_congestion_options(2)
```

placer_enable_enhanced_soft_blockages

Prevents cells already placed on soft blockages, at the beginning of placement, from being moved out of the soft blockages.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable affects only incremental coarse placement. It has no effect on synthesis or legalization.

Soft blockages prevent the coarse placer from placing cells on the blockage. They do not prevent synthesis or legalization placing cells. After an initial placement, synthesis may add cells on top of a soft blockage. If incremental coarse placement is later done on the design, it will move those new cells off the soft blockages. This may be undesirable.

Set this variable to true to prevent incremental coarse placement from moving cells out of soft blockages.

If this variable is set to true, cells already on soft blockages at the beginning of incremental coarse placement will be allowed, but not constrained, to stay there.

Cells not on soft blockages at the beginning of incremental coarse placement will always be prevented from moving onto the soft blockages during placement, irrespective of the value of this variable.

SEE ALSO

`create_placement_blockage(2)`

placer_enable_high_effort_congestion

Enables high-effort congestion removal during coarse placement.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables high-effort congestion removal during congestion-driven coarse placement when set to **true**.

Use this variable on congested designs when the default congestion effort is insufficient to remove all of the congestion. You may see longer runtimes as a side effect.

Use this feature in conjunction with any command that invokes congestion-driven placement, such as **place_opt -congestion** and **psynopt -congestion**.

For example, to enable this feature, enter

```
prompt> set placer_enable_high_effort_congestion true
```

SEE ALSO

```
place_opt(2)
psynopt(2)
```

placer_enable_redefined_blockage_behavior

Enables new behavior and features related to placement blockages.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable affects coarse placement when your design contains enhanced soft blockages, no-register blockages, buffer-only blockages, no-relative-placement blockages, and category blockages.

Set this variable to **true**, when:

- Your design contains any of the types of blockages previously listed, to prevent clumping of cells in the region outside the blockages.
- Your design contains more than one of the types of blockages previously listed, to ensure the correct interaction between the different types of blockages.
- You create buffer-only blockages using the **create_placement_blockage -buffer_only** command.
- You create category blockages using the **create_placement_blockage -category** command.

SEE ALSO

`create_placement_blockage(2)`

placer_gated_register_area_multiplier

Specifies the value of the multiplier used to generate automatic group bounds for gated clocks.

TYPE

float

DEFAULT

20

DESCRIPTION

The gated-clock method reduces a design's power consumption by controlling the switching activity from the clock signal to flip-flop registers. The automatic bound function works only for the gated-clock elements created by Power Compiler. The **placer_gated_register_area_multiplier** variable does not affect your design unless you use the clock-gating functionality of Power Compiler.

To reduce clock skew, place the clock-gating element and the flip-flops it controls close together in your design. The tool automatically creates a floating group bound for each clock-gating element and the flip-flops the element drives. This group bound helps place the flip-flops closer to the clock-gating element.

The size of the group bound is equal to the square root of the product of the total area of flip-flop elements the gate controls within the group bound and the value of the **placer_gated_register_area_multiplier** variable.

For example, if a clock-gating element drives five flip-flops and the total area of these six cells is 45, the size of the group bound is sqrt(20 x 45) = 30.

SEE ALSO

`physopt_check_site_array_overlap(3)`

placer_max_allowed_timing_depth

Specifies the maximum timing graph depth allowed by the placer.

TYPE

integer

DEFAULT

20000

DESCRIPTION

The placer cannot process extremely deep timing paths, and will check for the presence of such paths. This variable controls the depth limit for the checking.

When running the placer, no path may be deeper than this limit. This applies even if the path is unconstrained.

If this variable is set above 32767, however, two things will happen:

1) Depth checking will be disabled.

2) Multicore operation will be disabled.

Be aware that increasing this limit may cause the process to abort due to insufficient stack space. Always set the process stack size to a high value in your system command shell when using this option.

An alternative to changing this limit is to use set_disable_timing to break very long paths into multiple segments. The paths must be broken up so than no segment (even if unconstrained) is longer than the limit.

SEE ALSO

placer_max_cell_density_threshold

Enables a mode of coarse placement in which cells can clump together.

TYPE

float

DEFAULT

-1

DESCRIPTION

This variable enables a mode of coarse placement in which cells are not distributed evenly across the surface of the chip, but are allowed to clump together. The value you specify sets the threshold of how tightly the cells are allowed to clump. A value of 1.0 allows no gaps between cells.

A reasonable value is one that is above the background utilization of your design but below 1.0. For example, if your background utilization is 40%, or 0.4, a reasonable value for this variable is a value between 0.4 and 1.0. The higher the value, the more tightly the cells clump together.

You can use this feature in conjunction with any command that invokes coarse placement, such as **create_fp_placement**, **create_placement**, and **place_opt**.

For example, to set the threshold to 0.7, enter

```
prompt> set placer_max_cell_density_threshold 0.7
```

SEE ALSO

```
create_fp_placement(2)
create_placement(2)
place_opt(2)
```

placer_max_parallel_computations

Overrides the value set by the **set_host_options -max_cores** command for running the placement engine in multicore mode.

TYPE

integer

DEFAULT

0

GROUP

placer variables

DESCRIPTION

This variable can reduce the number of cores used for running the placement engine when you have previously enabled multicore processing by running the **set_host_options -max_cores** command.

If the variable is set to 0 (the default value), the maximum number of cores used by the placement engine is specified by the **set_host_options -max_cores** command.

If the variable is set to 1, the placement engine runs in single-core mode, regardless of the number of cores specified by the **set_host_options -max_cores** command.

If the variable is set to a value greater than 1 and smaller than the value specified by the **set_host_options -max_cores** command, the placement engine ignores the value set by the **set_host_options** command and runs in multicore mode with the number of cores specified by the variable.

If the variable is set to a value greater than the value specified by the **set_host_options -max_cores** command, this variable is ignored.

This variable should not be used for normal operation and is primarily intended for diagnostic purposes.

EXAMPLES

The following example enables multicore processing with a maximum of four cores for all multicore functions.

```
prompt> set_host_options -max_cores 4
```

The following example enables multicore processing with a maximum of two cores for the placement engine and a maximum of four cores for all other multicore functions.

```
set_host_options -max_cores 4
set placer_max_parallel_computations 2
```

The following example enables singles-core processing for the placement engine and a maximum of four cores for all other multicore functions.

```
set_host_options -max_cores 4
set placer_max_parallel_computations 1
```

The following example enables multicore processing with a maximum of two cores for all multicore functions. The **placer_max_parallel_computations** variable is ignored because its value is greater than the value set the **set_host_options** command.

```
set_host_options -max_cores 2
set placer_max_parallel_computations 4
```

SEE ALSO

```
set_host_options(2)
```

placer_reduce_high_density_regions

Enables high-effort high-density-spot reduction during coarse placement.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables high-effort high-density-spot reduction during coarse placement when set to **true**.

Use this variable on designs that exhibit high-density areas around macros or blockages when the default coarse placement effort is insufficient to remove all these high-density areas. You might see longer runtimes as a side effect (coarse placement runtime is expected to double). This feature is not yet wire-length or timing aware, so wire-length and timing might degrade as a result of using it.

To enable this feature, enter

```
prompt> set placer_reduce_high_density_regions true
```

SEE ALSO

```
create_placement(2)
refine_placement(2)
place_opt(2)
place_opt_feasibility(2)
clock_opt(2)
clock_opt_feasibility(2)
psynopt(2)
```

placer_show_zroutegr_output

Prints out Zroute global routing information during congestion-driven placement.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether global routing information is output during congestion-driven placement when the Zroute global router is used.

By default (**false**), the global routing information is not output.

When you set this variable to **true**, the global routing information is output when using Zroute global routing for congestion-driven placement.

SEE ALSO

```
create_placement(2)
place_opt(2)
```

plan_group_attributes

Contains attributes related to plan group.

DESCRIPTION

Contains attributes related to plan group.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class plan_group -application**, the definition of attributes can be listed.

Plan Group Attributes

aspect_ratio

Specifies the **height**:**width** ratio of a plan group.

The data type of **aspect_ratio** is double.

This attribute is read-only.

bbox

Specifies the bounding-box of a plan group. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ll

Specifies the lower-left corner of the bounding-box of a plan group.

The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ll** of a plan group, by accessing the first element of its **bbox**.

The data type of **bbox_ll** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a plan group.

The data type of **bbox_llx** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a plan group.

The data type of **bbox_lly** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ur

Specifies the upper-right corner of the bounding-box of a plan group.

The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ur** of a plan group, by accessing the second element of its **bbox**.

The data type of **bbox_ur** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a plan group.

The data type of **bbox_urx** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a plan group.

The data type of **bbox_ury** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bottom_padding

Specifies bottom-side width of interior paddings.

The data type of **bottom_padding** is double.

This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

bottom_padding_external

Specifies bottom-side width of exterior paddings.

The data type of **bottom_padding_external** is double.

This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

bottom_shielding

Specifies bottom-side widths of signal shielding inside a plan group.

The data type of **bottom_shielding** is string.

This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

bottom_shielding_external

Specifies bottom-side widths of signal shielding outside a plan group.

The data type of **bottom_shielding_external** is string.

This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

color

Specifies color to draw a plan group and its associated instances.

The data type of **color** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

is_fixed

Specifies that a plan group is in a fixed location, and the shaping will ignore it.

The data type of **is_fixed** is boolean.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

is_on_demand_netlist

Specifies that a plan group contains only the interface netlist.

The data type of **is_on_demand_netlist** is boolean.

This attribute is read-only.

left_padding

Specifies left-side width of interior paddings.

The data type of **left_padding** is double.

This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

left_padding_external

Specifies left-side width of exterior paddings.

The data type of **left_padding_external** is double.

This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

left_shielding

Specifies left-side widths of signal shielding inside a plan group.

The data type of **left_shielding** is string.

This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

left_shielding_external

Specifies left-side widths of signal shielding outside a plan group.

The data type of **left_shielding_external** is string.

This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

logic_cell

Specifies name of the hierarchical cell associated with a plan group.

The data type of **logic_cell** is string.

This attribute is read-only.

name

Specifies name of a plan group object.

The data type of **name** is string.

This attribute is read-only.

number_of_hard_macro

Specifies number of hard macro cells inside a plan group.

The data type of **number_of_hard_macro** is integer.

This attribute is read-only.

number_of_pin

Specifies number of pins on the hierarchical cell associated with a plan group.

The data type of **number_of_pin** is integer.

This attribute is read-only.

number_of_standard_cell

Specifies number of standard cells inside a plan group.

The data type of **number_of_standard_cell** is integer.

This attribute is read-only.

number_of_odl_standard_cell

Specifies number of standard cells inside an ODL plan group.

The data type of **number_of_odl_standard_cell** is integer.

This attribute is read-only.

object_class

Specifies object class name of a plan group, which is **plan_group**.

The data type of **object_class** is string.

This attribute is read-only.

points

Specifies point list of a plan group's boundary.

The data type of **points** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

right_padding

Specifies right-side width of interior paddings.

The data type of **right_padding** is double.

This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

right_padding_external

Specifies right-side width of exterior paddings.

The data type of **right_padding_external** is double.

This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

right_shielding

Specifies right-side widths of signal shielding inside a plan group.

The data type of **right_shielding** is string.

This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

right_shielding_external

Specifies right-side widths of signal shielding outside a plan group.

The data type of **right_shielding_external** is string.

This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

target_utilization

Specified target utilization set on a plan group.

The data type of **target_utilization** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

top_padding

Specifies top-side width of interior paddings.

The data type of **top_padding** is double.

This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

top_padding_external

Specifies top-side width of exterior paddings.

The data type of **top_padding_external** is double.

This attribute is read-only. You can use **create_fp_plan_group_padding** to set its value.

top_shielding

Specifies top-side widths of signal shielding inside a plan group.

The data type of **top_shielding** is string.

This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

top_shielding_external

Specifies top-side widths of signal shielding outside a plan group.

The data type of **top_shielding_external** is string.

This attribute is read-only. You can use **create_fp_block_shielding** to set its value.

utilization

Specifies the ratio of total area size of associated instances to the area of a plan group.

The data type of **utilization** is double.

This attribute is read-only.

odl_utilization

Specifies the ratio of total area size of associated instances to the area of an ODL plan group.

The data type of **odl_utilization** is float.

This attribute is read-only.

is_mim

Specifies that a plan group is multiply instantiated module (MIM).

The data type of **is_mim** is Boolean.

This attribute is read-only.

is_mim_master_instance

Specifies that a plan group is a master instance among a set of multiply instantiated modules (MIMs).

The data type of **is_mim_master_instance** is Boolean.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

mim_master_name

Specifies the name of the master for a multiply instantiated module (MIM) plan group.

The data type of **mim_master_name** is string.

This attribute is read-only.

mim_orientation

Specifies the orientation of a MIM plan group.

The allowable orientations are N, S, FN, FS.

The data type of **mim_orientation** is string.

This attribute is read-only.

SEE ALSO

```
create_fp_block_shielding(2)
create_fp_plan_group_padding(2)
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

pns_commit_lower_layer_first

Defines the order that power straps are committed during Power Network Synthesis (PNS).

TYPE

Boolean

DEFAULT

false

GROUP

pns_variables

DESCRIPTION

By default, IC Compiler commits higher metal layer power straps before lower metal layer power straps during power network synthesis. This ensures the integrity of the higher metal layer mesh.

In certain cases, the vias dropped from the higher metal layers might block the routing on lower metal layers. The **pns_commit_lower_layer_first** variable directs IC Compiler to create lower metal layer straps before higher metal layer straps to prevent via blockages.

Note that in template-based PNS, this variable affects only straps within the same strategy. In other words, IC Compiler first commits straps on lower layers within the same strategy, but not within other strategies.

SEE ALSO

`compile_power_plan(2)`

pns_do_not_connect_pin

Determines whether Power Network Synthesis (PNS) drops via to macro pins when the power straps route over the macros.

TYPE

Boolean

DEFAULT

false

GROUP

pns_variables

DESCRIPTION

By default, PNS drops vias to connect pins inside macros when the power straps route over the macros. However, these vias might block other routes that would cause a DRC violation.

You can use the **pns_do_not_connect_pin** variable to create the power mesh without dropping vias. Set the **pns_do_not_connect_pin** to true, generate the power mesh on top of macros by using PNS, and specify the **preroute_instances** command to connect the macro pins to the power mesh.

SEE ALSO

`preroute_instances(2)`

pns_do_not_generate_pin

Defines whether IC Compiler generates new pins at the design boundary when you extend straps during Power Network Synthesis (PNS).

TYPE

Boolean

DEFAULT

false

GROUP

pns_variables

DESCRIPTION

By default, when you extend a power strap to the design boundary by using the **set_power_plan_strategy** command, IC Compiler generates a new pin at the design boundary. This is the desired behavior for most hierarchical flows.

In certain cases, pins might already exist at the design boundary after pushing down the power mesh from the top level, and no new pins are required. You can set **pns_do_not_generate_pin** to true to disable pin creation in this case.

SEE ALSO

`set_power_plan_strategy(2)`

pns_ignore_cell_boundary

Allow straps to route on the design boundary.

TYPE

Boolean

DEFAULT

false

GROUP

pns_variables

DESCRIPTION

In PNS,

By default, Power Network Synthesis (PNS) does not route power straps outside the design boundary.

You can route power straps with the strap centerline on the boundary by setting **pns_ignore_cell_boundary** to true. This variable applies only to power straps that overlap the boundary. Power straps that are inside the boundary are not affected. After setting this variable to false, subsequent steps in the PNS flow are not affected.

SEE ALSO

`compile_power_plan(2)`

pns_quick_drc

Speeds up power network synthesis runtime when there are a large number of straps (often more than one million) to be created.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, power network synthesis performs design rule checking (DRC) to perform accurate wire cutting and snapping during synthesis and when committing the power network. However, when you create a large number of straps and the supplied spacing is correct, the checking during the synthesis stage can cause long runtimes.

To speed up the runtime, set the **pns_quick_drc** variable to **true**. Note that DRC is still performed when committing the PG wires in the layout, even when this variable is set to **true**. Before setting this variable to **true**, you should ensure that the spacing defined in template is correct; otherwise, wire cutting and trimming is not performed and wires might not be created successfully.

SEE ALSO

`compile_power_plan(2)`

port_attributes

Contains attributes that can be placed on a port.

DESCRIPTION

Contains attributes that can be placed on a port.

There are a number of commands used to set attributes. Most attributes, however, can be set with the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute. Otherwise, use **set_attribute**. If an attribute is "read-only," the user cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, refer to the manual pages of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Port Attributes

bus_name

If the port is bussed, returns the full hierarchical name of the port bus. This attribute is **read only** and cannot be set by the user.

connection_class

A string that specifies the connection class label to be attached to a port or to a list of ports. **compile**, **insert_pads**, and **insert_dft** will connect only those loads and drivers that have the same connection class label. The labels must match those in the library of components for the design, and must be separated by a space. The labels *universal* and *default* are reserve.d *universal* indicates that the port can connect with any other load or driver. *default* is assigned to any ports that do not have a connection class already assigned. Set with **set_connection_class**.

dont_touch_network

When a design is optimized, **compile** assigns **dont_touch** attributes to all cells and nets in the transitive fanout of **dont_touch_network** clock objects. The **dont_touch** assignment stops at the boundary of storage elements. An element is recognized as storage only if it has setup or hold constraints. Set with **set_dont_touch_network**.

driven_by_logic_one

Specifies that input ports are driven by logic one. **compile** uses this information to create smaller designs. After optimization, a port connected to logic one usually does not drive anything inside the optimized design. Set with **set_logic_one**.

driven_by_logic_zero

Specifies that input ports are driven by logic zero. **compile** uses this information to create smaller designs. After optimization, a port connected to logic zero usually does not drive anything inside the optimized design. Set with **set_logic_zero**.

driving_cell_dont_scale

When *true*, the transition time on the port using the driving cell is not scaled. Otherwise the transition time will be scaled by operating condition factors. Set with **set_driving_cell**.

driving_cell_fall

A string that names a library cell from which to copy fall drive capability to be used in fall transition calculation for the port. Set with **set_driving_cell**.

driving_cell_from_pin_fall

A string that names the driving_cell_fall input pin to be used to find timing arc fall drive capability. Set with **set_driving_cell**.

driving_cell_from_pin_rise

A string that names the driving_cell_rise input pin to be used to find timing arc rise drive capability. Set with **set_driving_cell**.

driving_cell_library_fall

A string that names the library in which to find the **driving_cell_fall**. Set with **set_driving_cell**.

driving_cell_library_rise

A string that names the library in which to find the **driving_cell_rise**. Set with **set_driving_cell**.

driving_cell_multiply_by

A floating point value by which to multiply the transition time of the port marked with this attribute. Set with **set_driving_cell**.

driving_cell_pin_fall

A string that names the driving_cell_fall output pin to be used to find timing arc fall drive capability. Set with **set_driving_cell**.

driving_cell_pin_rise

A string that names the driving_cell_rise output pin to be used to find timing arc rise drive capability. Set with **set_driving_cell**.

driving_cell_rise

A string that names a library cell from which to copy rise drive capability to be used in rise transition calculation for the port. Set with **set_driving_cell**.

fall_drive

Specifies the drive value of high to low transition on input or inout ports. Set with **set_drive**.

fanout_load

Specifies the fanout load on output ports. Set with **set_fanout_load**.

is_bussed

Returns true if the port is part of a bussed port. This attribute is **read only** and cannot be set by the user.

load

Specifies the load value on ports. The total load on a net is the sum of all the loads on pins, ports, and wires associated with that net. Set with **set_load**.

max_capacitance

A floating point number that sets the maximum capacitance value for input, output, or bidirectional ports; or for designs. The units must be consistent with those of the technology library used during optimization. Set with **set_max_capacitance**.

max_fall_delay

A floating point value that specifies the maximum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_fanout

Specifies the maximum fanout load for the net connected to this port. **compile** ensures that the fanout load on this net is less than the specified value. Set with **set_max_fanout**.

max_rise_delay

A floating point value that specifies the maximum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_max_delay**.

max_time_borrow

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 technology library. Set with **set_max_time_borrow**.

max_transition

Specifies the maximum transition time for the net connected to this port. **compile** ensures that the transition time on this net is less than the specified value. Set with **set_max_transition**.

min_capacitance

A floating point number that sets the minimum capacitance value for input or bidirectional ports. The units must be consistent with those of the technology library used during optimization. Set with **set_min_capacitance**.

min_fall_delay

A floating point value that specifies the minimum falling delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

min_rise_delay

A floating point value that specifies the minimum rising delay on ports, clocks, pins, cells, or on paths between such objects. Set with **set_min_delay**.

model_drive

A non-negative floating point number that specifies the estimated drive value on ports in terms of standard drives of the current technology library. Set with **set_model_drive**.

model_load

A non-negative floating point number that specifies the estimated load value on ports in terms of standard loads of the current technology library. Set with **set_model_load**.

op_used_in_normal_op

Specifies that a scan-out port is also used in normal operation (system mode). This attribute is used by the **insert_dft** command.

output_not_used

Determines that an output port is unconnected. Used by **compile** to create smaller designs because the logic that drives an unconnected output port might not need to be maintained. After a design with an unconnected output port is compiled, the port is usually not driven by anything inside the design. Set with **set_unconnected**.

pad_location

A string value that specifies the Xilinx pad location (pin number) to be assigned to a port. Setting a **pad_location** attribute on a port causes the Synopsys XNF writer to indicate in the XNF netlist that this port has the pad location given by the value of the **pad_location** attribute. No checks are performed to verify that the specified location is valid; for valid pad locations, refer to the Xilinx *XC4000 Databook*. Set with **set_attribute**.

port_direction

Returns the direction of a port. The value can be **in**, **out**, **inout**, or **unknown**. This attribute cannot be set by the user. You can also use the **direction** attribute to get the direction of the port.

rise_drive

Specifies the drive value of low to high transition on input or inout ports. Set with **set_drive**.

signal_index

Used to enumerate different ports with the same signal type (for example, scan-in ports for a design with multiple scan chains). Set with **set_signal_type**.

signal_type

Used to indicate that a port is of a "special" type, such as a "clocked_on_also" port in a master/slave clocking scheme, or a "test_scan_in" pin for scan-test circuitry. Set with **set_signal_type**.

static_probability

A floating point number that specifies the percentage of time that the signal is in the logic 1 state. This information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 0.5, indicating that the signal is in the logic 1 state half the time. Set with **set_switching_activity**.

test_dont_fault

If set, ports are not faulted during test pattern generation. If no command options are specified, this attribute is set for both "stuck-at-0" and "stuck-at-1" faults. Set with **set_test_dont_fault**.

test_hold

Specifies a fixed, constant logic value at a port during test generation. Set with **set_test_hold**.

test_isolate

Indicates that the specified sequential cells, pins, or ports are to be logically isolated and considered untestable during test design rule checking by **check_test**. When this attribute is set on a cell, it is also placed on all pins of that cell. Do not set this attribute on a hierarchical cell. Use **report_test -assertions** for a report on isolated objects. Set with **set_test_isolate**.

Note: Setting this attribute suppresses the warning messages associated with the isolated objects.

toggle_rate

A positive floating point number that specifies the toggle rate; that is, the number of zero-to-one and one-to-zero transitions within a library time unit period. This information is used by **report_power**. If this attribute is not set, **report_power** will use the default value of 2*(static_probability)(1 - static_probability). The default will be scaled by any associated clock signal (if one is available). Set with **set_switching_activity**.

true_delay_case_analysis

Specifies a value to set all or part of an input vector for **report_timing -true** and **report_timing -justify**. Allowed values are *0*, *1*, *r* (rise, X to 1), and *f* (fall, X to 0). Set with **set_true_delay_case_analysis**.

is_diode

Tells if the port is a diode port.

is_bus

Tells if the port is a bus port.

This attribute is read-only and cannot be modified.

clock_latency_fall_max

A float attribute representing the user-specified clock network latency for max fall. This attribute is only defined on those ports where **set_clock_latency** has been directly set.

clock_latency_fall_min

A float attribute representing the user-specified clock network latency for min fall. This attribute is only defined on those ports where **set_clock_latency** has been directly set.

clock_latency_rise_max

A float attribute representing the user-specified clock network latency for max rise. This attribute is only defined on those ports where **set_clock_latency** has been directly set.

clock_latency_rise_min

A float attribute representing the user-specified clock network latency for min rise. This attribute is only defined on those ports where **set_clock_latency** has been directly set.

hold_uncertainty

Specifies a negative uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_uncertainty -hold**.

setup_uncertainty

Specifies a positive uncertainty from the edges of the ideal clock waveform. Affects all sequential cells in the transitive fanout of this port. Set with **set_clock_uncertainty -setup**.

pg_pin_weight

A float attribute representing the PG pin weight. It should be between 0.1 and 25.0. And the fraction precision is 0.1, e.g. 0.11 will be cut off as 0.1.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)
```

port_complement_naming_style

Defines the convention the **compile** command uses to rename ports complemented as a result of using the **set_boundary_optimization** command.

TYPE

string

DEFAULT

%s_BAR

GROUP

compile_variables

DESCRIPTION

This variable defines the convention the **compile** command uses to rename ports complemented as a result of using the **set_boundary_optimization** command.

The variable string must contain one occurrence of %s (percent s). When **compile** generates a new port name, %s is replaced with the original name of the port. If this does not create a unique port name within the cell, the smallest possible integer that makes the name unique is appended to the end of the name (for example, X_BAR_0, X_BAR_1, and so on).

To determine the current value of this variable, use the **printvar port_complement_naming_style** command. For a list of all compile variables and their current values, use **print_variable_group compile**.

SEE ALSO

power_attributes

Lists the predefined Power Compiler attributes.

DESCRIPTION

Attributes are properties assigned to objects such as nets, cells, pins and designs, and describe design features to be considered during optimization. A subset of attributes are specific to Power Compiler; they are only available when a Power Compiler license exists.

The Power Compiler attributes are "read-only": they cannot be set by the user. To determine the value of an attribute, use the **get_attribute** command. For information on all attributes, refer to the **attributes** manual page.

The Power Compiler attributes are grouped into the following categories:

- `cell`
- `pin`
- `design`

Definitions for these attributes are provided in the subsections that follow.

Cell Attributes

is_clock_gate

true if the cell is a clock gate. If the clock gating logic is encapsulated in a hierarchical clock gate wrapper, this attribute will only return *true* when applied to the hierarchical instance.

is_icg

true if the cell is an integrated clock gate (ICG). This attribute can only return *true* when applied to leaf cells. If the ICG cell is encapsulated in a hierarchical clock gate wrapper, this attribute will return *false* when applied to the hierarchical instance.

is_gicg

true if the cell is a generic integrated clock gate (GICG). Since GICGs cannot be encapsulated in a hierarchical clock gate wrapper, this attribute can only return *true* when applied to leaf cells.

is_latch_based_clock_gate

true if the cell is a latch-based clock gate.

is_latch_free_clock_gate

true if the cell is a latch-free clock gate.

is_positive_edge_clock_gate

true if the cell is a positive edge clock gate.

is_negative_edge_clock_gate

true if the cell is a negative edge clock gate.

clock_gate_has_precontrol

true if the cell is a clock gate with (pre-latch) control point.

clock_gate_has_postcontrol

true if the cell is a clock gate with (post-latch) control point.

clock_gate_has_observation

true if the cell is a clock gate with observation point.

is_clock_gated

true if the cell is a clock gated register or clock gate.

clock_gating_depth

The number of clock gates on the clock path to this cell; -1 if the cell is not a clock gate or register.

clock_gate_level

The number of clock gates on the longest clock branch in the fanout of this cell; -1 if not a clock gate.

clock_gate_fanout

The number of registers and clock gates in the direct fanout of the clock gate; -1 if not a clock gate.

clock_gate_register_fanout

The number of registers in the direct fanout of the clock gate; -1 if not a clock gate.

clock_gate_multi_stage_fanout

The number of clock gates in the direct fanout of the clock gate; -1 if not a clock gate.

clock_gate_transitive_register_fanout

The number of register in the transitive fanout of the clock gate; -1 if not a clock gate.

clock_gate_module_fanout

The number of modules in the local fanout of the clock gate; -1 if not a clock gate.

is_operand_isolator

true if the cell is an operand isolation cell.

is_isolated_operator

true if the cell is an operator that was isolated with operand isolation.

operand_isolation_style

Stores the operand isolation style of the isolation cell or isolated operator.

Pin Attributes

is_clock_gate_enable_pin

true if the pin is a clock gate enable input.

is_clock_gate_clock_pin

true if the pin is a clock gate clock input.

is_clock_gate_output_pin

true if the pin is a gated-clock output of a clock gate.

is_clock_gate_test_pin

true if the pin is a clock gate scan-enable or test-mode input.

is_clock_gate_observation_pin

true if the pin is a clock gate observation point.

is_operand_isolation_control_pin

true if the pin is the control pin of an operand isolation cell.

is_operand_isolation_data_pin

true if the pin is the data input of an operand isolation cell.

is_operand_isolation_output_pin

true if the pin is the data output of an operand isolation cell.

"Design Attributes

is_clock_gating_design

true if the design is a clock gating design.

is_clock_gating_observability_design

true if the design is a clock gating observability design.

SEE ALSO

`get_attribute(2)`
`attributes(3)`

power_cg_all_registers

Specifies to the **insert_clock_gating** command whether to clock gate all registers, including those that do not meet the necessary requirements.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable specifies whether to clock gate all registers, including those that do not meet the necessary requirements. If this variable is set to **false** (the default value), registers that do not meet the setup, width, or enable condition are not considered for clock gating unless they are explicitly included with the **set_clock_gating_registers** command. When set to **true**, a redundant clock gate is inserted for these registers. This can be useful for clock tree balancing. If necessary, the redundant clock gates are duplicated to meet the max_fanout constraint. Note that the minimum_bitwidth constraint is not honored for the redundant clock gated registers.

To determine the current value of this variable, use the **printvar power_cg_all_registers** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

power_cg_balance_stages

Controls whether gate stage balancing is on or off during **compile [-incremental_mapping] -gate_clock** or **compile_ultra [-incremental_mapping] -gate_clock**.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable reduces the unevenness in the number of stages of clock gates feeding different register banks. When this variable is set to **true**, the **compile -gate_clock** or **compile_ultra -gate_clock** command reconfigures the different stages of clock gates so that there is exactly the same number of clock gates in each path from the clock root to the clock pin of all gatable register banks. It is necessary to define clocks using the **create_clock** command for each clock network where clock gates need to be reconfigured.

Only the tool-inserted clock gates and integrated clock gating (ICG) cells are considered during stage balancing.

A clock-gating cell is not modified or removed if it or its parent hierarchical cell is marked as **dont_touch** with the **set_dont_touch** command; it will not be affected by clock gate stage balancing.

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

SEE ALSO

```
report_clock_gating(2)
power_cg_reconfig_stages(3)
```

power_cg_cell_naming_style

Specifies the naming style for clock-gating cells created during **insert_clock_gating**.

TYPE

string

DEFAULT

""

GROUP

power

DESCRIPTION

This variable determines the way that clock gating cells are named.

This variable must be set before issuing the **insert_clock_gating** command.

This variable can contain any string in addition to the tokens listed below. The tokens are replaced by the appropriate value during clock gating and other strings are retained without changes. For example:

```
set power_cg_cell_naming_style \
   "prefix_%c_%e_midfix_%r_%R_%d_suffix"

  %c - clock name
  %n - immediate enable signal name
  %r - first gated register bank name (or module name for module
       clock gates and not applicable for factored clock gates)
  %R - all gated register banks sorted alphabetically (not applicable
       for module or factored clock gates)
  %d - index for splitting/name clash resolution
```

Only simple (non-hierarchical) names are used for all object names.

If there is no occurrence of "%d" in **power_cg_cell_naming_style**, the tool assumes a %d at the end. The following are examples of the variable usage:

```
prompt> set power_cg_cell_naming_style "clk_gate_%r_%d
```

```
prompt> set power_cg_cell_naming_style "clock_gate_cell_%c_%r_%d_name
```

SEE ALSO

power_cg_gated_clock_net_naming_style(3)
power_cg_module_naming_style(3)

power_cg_derive_related_clock

Derives the clock domain relationship between registers from the hierarchical context.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

For latch-free clock gate insertion with the **insert_clock_gating** command, the setup condition must be met before a bank can be clock gated. This implies that the enable condition of the bank must be combinationally dependent on nets that are known to be synchronous (such as, belong to the same clock domain) with the registers of the bank that are being clock gated.

The clock domain of nets is determined by the registers withing the module (subdesign) that drive the nets and the clock relationship specified for ports with the **set_input_delay** command. By default, the clock domain relationship is analyzed within the module. This is to ensure that clock gating on subdesigns is context independent. If a module is instantiated from a design different than the current design, the clock-gating result will still be correct.

If the non-combinational driver (register or top-level port) of an input to a module exists outside of that module, the clock domain of that input is considered unknown and no latch-free clock gating can be performed with enable conditions that depend on that particular input.

The setup condition can be relaxed to perform context specific analysis of the clock domain relationship. This is achieved by setting the **power_cg_derive_related_clock** variable to **true**. In that case, the clock domain of any net will be derived from the non-combinational drivers in the (hierarchical) fanin of the net.

To determine the current value of this variable, use the **printvar power_cg_derive_related_clock** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

power_cg_designware

Performs clock gating on DesignWare sequential components in the design.

The **power_cg_designware** variable will be obsolete in a future release. Clock gating insertion with **compile_ultra -gate_clock** automatically inserts clock gates in DesignWare modules.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

The use of the **power_cg_designware** variable will be obsolete in a future release. Clock gating insertion using the **compile_ultra** command with the **-gate_clock** option automatically inserts clock gates in DesignWare modules.

The **power_cg_designware** variable instructs the **compile** command to invoke DesignWare clock gating. During the compile stage, all DesignWare sequential components are clock gated if there is a clock gating opportunity.

A Clock Gate Insertion Report is generated after Implementation Selection. You can use the **insert_dft** command to connect test ports inside DesignWare components. The **report_clock_gating** command can be used to get more details about the clock gating result.

The following example shows a TCL script invoking DesignWare clock gating:

```
prompt> set power_cg_designware true
prompt> analyze -f verilog DW_cntr_gray_inst.v
prompt> elaborate DW_cntr_gray_inst
prompt> create_clock -period 5 [get_ports inst_clk]
prompt> compile
```

```
prompt> report_clock_gating
```

SEE ALSO

power_cg_enable_alternative_algorithm

Specifies to the **insert_clock_gating**, **compile-gate_clock** and **compile_ultra -gate_clock** commands whether to use an alternative algorithm to find gatable registers.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

Depending on the design, clock gating can have a long runtime. When set to **true**, **power_cg_enable_alternative_algorithm** changes the clock gating algorithm. The alternative algorithm that is enabled often reduces the runtime for clock gating, especially for larger designs, but might increase the number of clock gates that are created compared to the default algorithm. The number of gated registers may vary.

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

SEE ALSO

power_cg_flatten

Specifies to different **ungroup** commands whether or not to flatten Synopsys clock-gating cells.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable specifies to different **ungroup** commands whether or not to flatten Synopsys clock-gating cells. The list of commands include **ungroup**, **compile -ungroup**, **optimize_registers -ungroup**, **balance_registers**, and **ungroup**.

If this variable is set to **false** (the default value), the clock-gating cells are not flattened during any ungroup step. To flatten the clock gating cells, set the value to **true** before using the **ungroup** command or any other ungrouping steps listed above.

In normal usage, ungrouping the clock gates is not recommended. Ungrouping the clock gates before compile could have serious side effects. For example, ungrouped clock gates cannot be mapped to integrated clock-gating cells. Power Compiler commands such as **report_clock_gating**, **remove_clock_gating**, and **rewire_clock_gating** assume that the clock gates have a hierarchy of their own. Also, the tool's placement of clock gates and registers may not be optimal when clock gating cells are flattened. Flattened clock gates are supported when using integrated clock-gating cells, provided the flattening is done only after **compile**.

To determine the current value of this variable, use the **printvar power_cg_flatten** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

`ungroup(2)`

power_cg_gated_clock_net_naming_style

Specifies the naming style for gated clock nets created during **insert_clock_gating**.

TYPE

string

DEFAULT

""

GROUP

power

DESCRIPTION

This variable determines the way gated clock nets are named.

This variable must be set before issuing the **insert_clock_gating** command.

This variable can contain any string in addition to the different tokens listed below. The tokens are replaced by the appropriate value during clock gating. For example:

```
set power_cg_gated_clock_net_naming_style \
   "prefix_%c_%n_%g_%d_suffix'

  %c - original clock
  %n - immediate enable signal name
  %g - clock gate (instance) name
  %d - index for splitting/name clash resolution
```

Only simple (non-hierarchical) names are used for all object names.

If there is no occurrence of "%d" in **power_cg_gated_clock_net_naming_style**, the tool assumes a %d at the end. The following is an example of the variable usage:

```
prompt> set power_cg_gated_clock_net_naming_style "gated_%c_%d"
```

SEE ALSO

power_cg_cell_naming_style(3)
power_cg_gated_clock_net_naming_style(3)
power_cg_module_naming_style(3)

power_cg_ignore_setup_condition

Ignores the setup condition for latch-free clock gating.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable ignores the setup condition for latch-free clock gating. For latch-free clock gate insertion with the **insert_clock_gating** command, the setup condition must be met before a bank can be clock gated. This implies that the enable condition of the bank must be combinationally dependent on nets that are known to be synchronous (belong to the same clock domain) with the registers of the bank that are being clock gated.

The setup condition can be ignored during latch-free clock gating. This is achieved by setting the **power_cg_ignore_setup_condition** variable to **true**. This is generally not safe, since latch-free clock gating of a bank with an enable that is not synchronous to the registers of that bank can lead to problems on the clock net. In this case it is the responsibility of the designer to ensure that the result after clock gating is functionally correct.

To determine the current value of this variable, use the **printvar power_cg_ignore_setup_condition** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

power_cg_inherit_timing_exceptions

Specifies that during **compile -gate_clock** or **compile_ultra [-incr] -gate_clock**, timing exceptions defined on registers must be automatically inferred onto the enable pin of the clock gate that is gating these registers.

TYPE

Boolean

DEFAULT

false

GROUP

power

DESCRIPTION

If this variable is set before running the **compile -gate_clock** or **compile_ultra -gate_clock**, then clock gate insertion tries to inherit the timing exceptions of the gated registers to the clock gate enable pin. The following conditions apply:

- Only cell level exceptions are inherited. This means that all synchronous pins of a register must have the same set of timing exceptions before this can be inferred onto a clock gate.
- If a set of registers with the same enable conditions has different timing exceptions among them, then this is split into a smaller set of registers with homogeneous exceptions before clock gate insertion. This might create small banks that fall below the minimum bit width specified for clock gating and so they may remain ungated.

SEE ALSO

power_cg_iscgs_enable

Specifies the use of instance-specific clock-gating style for the **set_clock_gating_style** and **remove_clock_gating_style** commands, and clock gate insertion during compile.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable specifies whether or not to use instance-specific clock-gating style during clock gate insertion, as well as to enable commands or options related to instance-specific clock-gating style. When this variable is set to **true**, instance-specific clock-gating style is enabled.

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

SEE ALSO

power_cg_module_naming_style

Specifies the naming style for clock gating modules created during **insert_clock_gating**.

TYPE

string

DEFAULT

""

GROUP

power

DESCRIPTION

This variable determines the way clock gate modules (hierarchical designs created for clock gates during **insert_clock_gating**) are named.

This variable must be set before issuing the **insert_clock_gating** command.

This variable can contain any string in addition to the different tokens listed below. The tokens are replaced by the appropriate value during clock gating, and prefix, midfix, and suffix are examples of any constant strings that can be specified.

```
set power_cg_module_naming_style \
   "prefix_%e_%l_midfix_%p_%t_%d_suffix"

  %e - edge type (HIGH/LOW)
  %l - library name of ICG cell library (if using ICG cells) or
       concatenated target_library names
  %p - immediate parent module name
  %t - top module (current design) name
  %d - index used to resolve name clash
```

If there is no occurrence of "%d" in **power_cg_module_naming_style**, the tool assumes a %d at the end. The following are examples of the variable usage:

```
prompt> set power_cg_module_naming_style "SNPS_CLOCK_GATE_%e_%p
```

```
prompt> set power_cg_module_naming_style "clock_gate_module_%e_%t_%d
```

SEE ALSO

power_cg_cell_naming_style(3)
power_cg_gated_clock_net_naming_style(3)

power_cg_print_enable_conditions

Reports the enable conditions of registers and clock gates during clock gate insertion.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable prints the enable conditions of clock gates and registers when performing automatic RTL clock gate insertion with the **insert_clock_gating** command. The report is useful for debugging purposes and to achieve a better understanding of the design structure and functionality.

The enable condition of a register or clock gate is a combinational function of nets in the design. As such, enable conditions can be represented by Boolean expressions of nets. The enable condition of a register represents the states for which a clock signal must be passed to the register. The enable condition of a clock gate corresponds to the states for which a clock will be passed to the registers in the fanout of the clock gate. The tool utilizes the enable condition of the registers for clock gate insertion.

To enable reporting of the enable conditions during clock gating, set **power_cg_print_enable_conditions** to **true** before issuing the **insert_clock_gating** command.

To determine the current value of this variable, use the **printvar power_cg_print_enable_conditions** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

`power_cg_print_enable_conditions_max_terms(3)`

power_cg_print_enable_conditions_max_terms

Specifies the maximum number of product terms to be reported in the sum of product expansion of the enable condition.

TYPE

integer

DEFAULT

10

GROUP

power_variables

DESCRIPTION

This variable specifies the maximum number of product terms to be reported in the sum of product expansion of the enable condition. For debugging purposes and to achieve a better understanding of the design structure and functionality it can be useful to print the enable conditions of clock gates and registers when performing automatic RTL clock gate insertion with the **insert_clock_gating** command.

The enable conditions are reported as a sum of product expression. Since for complex expressions such a representation is known to grow very large, the maximum number of product terms is limited by the **power_cg_print_enable_conditions_max_terms** variable. By default, at most 10 product terms are printed. If the actual number of product terms exceeds this limit, the enable condition is reported as "?? (too many product terms)". If necessary, the number of product terms to be shown can be increased by setting **power_cg_print_enable_conditions_max_terms** to a larger value.

To determine the current value of this variable, use the **printvar power_cg_print_enable_conditions_max_terms** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

power_cg_print_enable_conditions(3)

power_cg_reconfig_stages

Controls the reconfiguration of multistage clock gates during **compile [-incremental_mapping] -gate_clock** or **compile_ultra [-incremental_mapping] -gate_clock**.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable controls whether reconfigurations are performed during **compile_ultra -gate_clock** or **compile -gate_clock**. The reconfigurations are performed only if the **power_cg_reconfig_stages** variable is set to **true**.

Multistage reconfiguration involves the addition or reduction of stages of clock gates. You specify the maximum permissible number of stages of clock gates in the design using the **set_clock_gating_style -num_stages** command. A clock gate stage is added when a common enable condition can be factored out of many clock gates without violating the maximum stages specified. A clock gate stage is reduced if the design has more stages than what is specified.

Also, it is necessary to define clocks using the **create_clock** command for each clock network where clock gates need to be reconfigured.

Only the tool-inserted clock gates and integrated clock-gating (ICG) cells can be reconfigured.

A clock-gating cell is not modified or removed if it or its parent hierarchical cell is marked **dont_touch** with the **set_dont_touch** command; it will not be affected by multistage reconfiguration.

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

SEE ALSO

```
report_clock_gating(2)
power_cg_balance_stages(3)
```

power_default_static_probability

Specifies the default static probability value.

TYPE

float

DEFAULT

0.5

GROUP

power_variables

DESCRIPTION

The following variables are used to determine the switching activity of non-user annotated nets that are driven by primary inputs or black box cells:

power_default_static_probability
power_default_toggle_rate
power_default_toggle_rate_type\fp

For other unannotated nets, the tool 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 types of nets:

- User annotated values are used, even when the net is partially annotated (for example, the static probability is annotated, but the toggle rate is not).
- 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 annotated, then the default toggle rate value is derived from the **power_default_toggle_rate_type** and **power_default_toggle_rate** values. If the value of the **power_default_toggle_rate_type** variable is **fastest_clock** then the toggle rate value is

```
dtr * fclk
```

- where fclk is the frequency of the related clock if specified by the **set_switching_activity** command, or the frequency of the fastest clock in the design. If the design has no clocks, then a value of 1.0 is used for fclk, and dtr is the value of the **power_default_toggle_rate** variable.
- If the value of **power_default_toggle_rate_type** is **absolute**, then the value of the **power_default_toggle_rate** variable is used as the toggle rate.

The value of **power_default_static_probability** must be between 0.0 and 1.0, both inclusive. The value of **power_default_toggle_rate** must 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** must be 0.0. If the value of **power_default_toggle_rate** is 0.0, then the value of **power_default_static_probability** must be either 0.0 or 1.0.

The default value of **power_default_toggle_rate** is 0.1. The default value of **power_default_static_probability** is 0.5. The value of **power_default_toggle_rate_type** can be either **fastest_clock** or **absolute**. The default value is **fastest_clock**.

SEE ALSO

```
set_switching_activity(2)
power_default_toggle_rate(3)
power_default_toggle_rate_type(3)
```

power_default_toggle_rate

Specifies the default toggle rate value.

TYPE

float

DEFAULT

0.1

GROUP

power_variables

DESCRIPTION

The following variables are used to determine the switching activity of non-user annotated nets that are driven by primary inputs or black box cells:

power_default_static_probability
power_default_toggle_rate
power_default_toggle_rate_type

For other unannotated nets, the tool 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 types of nets:

- User annotated values are used, even when the net is partially annotated (for example, the static probability is annotated, but the toggle rate is not).
- 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 annotated, then the default toggle rate value is derived from the **power_default_toggle_rate_type** and **power_default_toggle_rate** values. If the value of the **power_default_toggle_rate_type** variable is **fastest_clock** then the following is used for the toggle rate value:

dtr * fclk

- where fclk is the frequency of the related clock if specified by the **set_switching_activity** command, or the frequency of the fastest clock in the design. If the design has no clocks, then a value of 1.0 is used for fclk, and dtr is the value of the **power_default_toggle_rate** variable.
- If the value of **power_default_toggle_rate_type** is **absolute**, then the value of the **power_default_toggle_rate** variable is used as the toggle rate.

The value of **power_default_static_probability** must be between 0.0 and 1.0, both inclusive. The value of **power_default_toggle_rate** must 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** must be 0.0. If the value of **power_default_toggle_rate** is 0.0, then the value of **power_default_static_probability** must be either 0.0 or 1.0.

The default value of **power_default_toggle_rate** is 0.1. The default value of **power_default_static_probability** is 0.5. The value of **power_default_toggle_rate_type** can be either **fastest_clock** or **absolute**. The default value is **fastest_clock**.

SEE ALSO

```
set_switching_activity(2)
power_default_static_probability(3)
power_default_toggle_rate_type(3)
```

power_default_toggle_rate_type

Specifies the default toggle rate type.

TYPE

string

DEFAULT

fastest_clock

GROUP

power_variables

DESCRIPTION

The following variables are used to determine the switching activity of non-user annotated nets that are driven by primary inputs or black box cells:

```
power_default_static_probability
power_default_toggle_rate
power_default_toggle_rate_type
```

For other unannotated nets, the tool 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 types of nets:

- User annotated values are used, even when the net is partially annotated (for example, the static probability is annotated, but the toggle rate is not).
- 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 annotated, then the default toggle rate value is derived from the **power_default_toggle_rate_type** and **power_default_toggle_rate** values. If the value of the **power_default_toggle_rate_type** variable is **fastest_clock**, then the following is used for the toggle rate value:

```
dtr * fclk
```

where fclk is the frequency of the related clock if specified by the **set_switching_activity** command, or the frequency of the fastest clock in the design. If the design has no clocks, then a value of 1.0 is used for fclk, and dtr is the value of the **power_default_toggle_rate** variable.

- If the value of **power_default_toggle_rate_type** is **absolute**, then the value of the **power_default_toggle_rate** variable is used as the toggle rate.

The value of **power_default_static_probability** must be between 0.0 and 1.0, both inclusive. The value of **power_default_toggle_rate** must 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** must be 0.0. If the value of **power_default_toggle_rate** is 0.0, then the value of **power_default_static_probability** must be either 0.0 or 1.0.

The default value of power_default_toggle_rate variable is 0.1. The default value of power_default_static_probability variable is 0.5. The value of **power_default_toggle_rate_type** can be either **fastest_clock** or **absolute**. The default value is **fastest_clock**.

SEE ALSO

```
set_switching_activity(2)
power_default_static_probability(3)
power_default_toggle_rate(3)
```

power_do_not_size_icg_cells

Controls whether **compile** does not size the integrated clock-gating cells in a design to correct DRC violations because doing so may result in lower area and power.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

When this variable is set to **true**, **compile** does not size the integrated clock-gating cells in the design to correct DRC violations, because doing so may result in lower area and power when the integrated clock-gating cell is the last element in the clock tree and drives all gated registers.

If the clock tree synthesis (CTS) tool inserts buffers after the clock gating, the fanout of the integrated clock-gating cell is limited to the clock-tree buffers. While running **compile** before performing clock tree synthesis, this information is not available in the design. If you set **power_do_not_size_icg_cells** to **true**, **compile** ignores DRC violations for the integrated clock-gating cells, because the CTS tool would insert buffers at the output of the cell. Once your design netlist has CTS buffers, you can set this variable to **false** to enable **compile** to fix any DRC violation still existing for the integrated clock-gating cell.

To determine the current value of this variable, use the **printvar power_do_not_size_icg_cells** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

power_domain_attributes

Describes the attributes related to power domains.

DESCRIPTION

Describes the attributes related to power domains. These attributes are defined only in UPF mode.

You can use the **get_attribute** command to determine value of an attribute, and use the **report_attribute** command to get a report of all attributes on a specified power domain. To see all power domain attributes, use the **list_attribute -class power_domain -application** command.

Power Domain Attributes

cell_id

Specifies Milkyway design ID in which a power domain object is located.

This attribute is read-only.

name

Specifies name of a power domain object.

This attribute is read-only.

full_name

Specifies full name of a power domain object.

This attribute is read-only.

object_class

Specifies object class name of a power domain object, which is **power_domain**.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

This attribute is read-only.

within_block_abstraction

Specifies whether the power domain is part of the block abstraction.

The data type of **within_block_abstraction** is Boolean.

This attribute is read-only and cannot be modified.

within_ilm

Specifies whether the power domain is part of an ILM.

The data type of **within_ilm** is Boolean.

This attribute is read-only and cannot be modified.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
```

power_enable_clock_scaling

Enables or disables clock scaling for power analysis in IC Compiler.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When set to **true**, this variable enables IC Compiler to scale dynamic power number according to clock frequencies specified in the SDC and the **set_power_clock_scaling** command.

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

```
printvar power_enable_clock_scaling
```

SEE ALSO

```
set_power_clock_scaling(2)
printvar(2)
```

power_enable_datapath_gating

Enables or disables datapath gating technique for a design.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

This variable enables or disables datapath gating, which is a dynamic power optimization technique for a design. By default, this variable is **true** which enables datapath gating. To disable datapath gating, set this variable to **false**.

Datapath gating inserts isolation cells for operators that have Synopsys low-power DesignWare architectures present in the synthetic library file, dw_minpower.sldb. Datapath gating is also performed on all datapath blocks extracted during the High Level Optimization step. For datapath gating, the tool always uses the adaptive gating style.

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

```
prompt> printvar power_enable_datapath_gating.
```

For a list of power variables and their current values, type **print_variable_group power**.

SEE ALSO

power_enable_one_pass_power_gating

Enables the one-pass flow power gating. This variable is for use only in non-UPF mode.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable enables one-pass flow power gating when set to **true**. One-pass flow eliminates the need for an incremental compile, and simplifies the user interface.

The following are examples of the original power-gating flow and the one-pass flow:

- Original power-gating flow:

```
prompt> set_power_gating_style -type CLK_FREE \
        [get_cells lev1b_inst/*reg*]

prompt> set_power_gating_signal -power_pin_index 1 \
        [get_pin lev1b_inst/retain]

prompt> set_power_gating_signal -power_pin_index 2 \
        [get_pin lev1b_inst/shutdown]

prompt> set power_enable_power_gating true

prompt> compile

prompt> hookup_power_gating_ports

prompt> compile -incr
```

```
prompt> write -format verilog -hierarchy -output post_compile.v
```

One-pass power-gating flow:

```
prompt> set_power_gating_style -type CLK_FREE \
        [get_cells lev1b_inst/*reg*]

prompt> set_power_gating_signal -power_pin_index 1 \
        [get_pin lev1b_inst/retain]

prompt> set_power_gating_signal -power_pin_index 2 \
        [get_pin lev1b_inst/shutdown]

prompt> set power_enable_one_pass_power_gating true

prompt> hookup_power_gating_ports -port_naming_style \
        {lev1a_inst/retain lev1a_inst/shutdown} \
        -default_port_naming_style power_pin_%d

prompt> compile

prompt> write -format verilog -hierarchy -output post_compile.v
```

SEE ALSO

power_enable_power_gating

Enables the power-gating flow that allows the selected retention registers from the target library to be used to map sequential elements. This variable can be used only in non-UPF mode. In UPF mode, use UPF commands to enable the power-gating flow.

TYPE

Boolean

DEFAULT

false

GROUP

power

DESCRIPTION

The **power_enable_power_gating** variable enables and disables the power gating flow during **compile** and **physopt** command activity.

Retention registers are the registers that can save the values of the registers and restore them later. Retention registers might have different styles. The **power_gating_cell** cell-level attributes in target libraries are used to specify the styles.

To allow only the retention registers with certain types used for the specified sequential elements during **compile**, set the **power_enable_power_gating** variable to **true**.

The command requires a Power Compiler license during **compile**, otherwise Design Compiler cannot handle retention registers correctly. If the variable is **true**, it also prevents the **compile** and **physopt** commands from swapping the retention registers back to regular sequential elements.

SEE ALSO

power_fix_sdpd_annotation

Specifies whether user-annotated SDPD switching activity annotation is corrected before it is used.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

This variable specifies whether the tool modifies the user-annotated state-dependent and/or path-dependent (SDPD) switching activity to fix any inconsistencies before they are used. The accuracy of switching activity annotation, including the SDPD annotation, affects the accuracy of power calculation, and when this variable is set the tool checks the SDPD annotation for inconsistencies. The SDPD annotation is automatically modified to fix any inconsistencies found.

In most cases, some inconsistencies are created during normal switching activity flows, and the SDPD fixing step modifies the SDPD annotation slightly to improve the power estimation accuracy. For example, during SAIF generation using a simulator, the total of rise and fall toggle on a pin may be different, when perhaps an odd number of toggles are captured. In this case, the SDPD fixing step scales the SDPD toggle rate annotations so that the rise and fall totals are the same.

Setting this variable to **false** disables the SDPD fixing step.

Setting the **power_fix_sdpd_annotation_verbose** variable to **true** makes the SDPD fixing step issue verbose messages when user annotated switching activity is modified.

The following checks and modifications are performed during the SDPD fixing step:

- False states (states that always evaluate to 0, including the default state on cells whose other states cover all possible states) with non-zero state-dependent (SD) static probabilities have their static probability set to 0.0.
- Unannotated false states on cells with partially annotated SD static probabilities are automatically annotated with 0.0.
- Cells with fully annotated SD static probabilities have their static probabilities scaled so that they add up to 1.0.
- Cells with partially annotated SD static probabilities that add up to more than 1.0 have their static probabilities scaled down so that they add up to 1.0. On such cells, an SD static probability of 0.0 is set on the unannotated states.
- False states and arcs with non-zero state-dependent and/or path-dependent (SDPD) toggle rates have their toggle rate set to 0.0.
- Unannotated false states/arcs on pins with partially annotated SDPD toggle rates are automatically annotated with 0.0.
- Pins with fully annotated SDPD toggle rates have their toggle rates scaled so that they add up to the non-SDPD toggle rate of the pin (that is, the non-SDPD toggle rate on the net connected to the pin).
- Pins with partially annotated SDPD toggle rates that add up to more than the pin toggle rate have their SDPD toggle rates scaled down so that they add up to the non-SDPD toggle rate. On such pins, an SDPD toggle rate of 0.0 is set on the unannotated states and arcs.
- Pins with partially annotated SDPD toggle rates have their toggle rates scaled so that the totals of rise and fall SDPD toggle rates are equal.
- The sum of the SDPD toggle rates on pins with fully annotated SDPD toggle rate information is annotated as the non-SDPD pin toggle rate (that is, the total toggle rate of the net connected to the pin) if this was not previously annotated.

SEE ALSO

```
power_fix_sdpd_annotation_verbose(3)
power_sdpd_message_tolerance(3)
```

power_fix_sdpd_annotation_verbose

Specifies whether verbose messages are reported during the fixing of user-annotated SDPD switching activity.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When this variable is set to **true**, an information or warning message is reported for every modification (exceeding a tolerance criteria) to the user-annotated SDPD switching activity performed by the SDPD fixing step. The SDPD fixing step is enabled by the **power_fix_sdpd_annotation** variable. See the man page for **power_fix_sdpd_annotation** for more information on this step.

By default, the **power_fix_sdpd_annotation_verbose** verbose messages are not reported and a single message indicating that SDPD annotation is being modified is reported instead. Whether this variable is set to **true** or not, SDPD fixing messages are only reported if they exceed the tolerance criteria specified by the **power_sdpd_message_tolerance** variable.

SEE ALSO

```
power_fix_sdpd_annotation(3)
power_sdpd_message_tolerance(3)
```

power_hdlc_do_not_split_cg_cells

Specifies that the **insert_clock_gating** command will not split clock-gating cells to limit their fanout.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable, when set to **true** specifies that the **insert_clock_gating** command will not split clock-gating cells to limit their fanout. By default, **insert_clock_gating** splits clock-gating cells to limit their fanout. This activity is based on the value specified by the **set_clock_gating_style -max_fanout** command, whose default is unlimited. When **true**, **insert_clock_gating** does not split clock-gating cells, resulting in a netlist where all registers are gated by a single clock-gating cell if they share the same enable signal. Also, **insert_clock_gating** does not honor the value specified by **set_clock_gating_style -max_fanout**.

Set the variable to **true** if your clock-tree synthesis (CTS) tool inserts buffers after the clock-gating cell. In this case, the fanout of the clock-gating cell is limited to the buffers inserted by the CTS tool. This information is not available in the design while the **insert_clock_gating** command is performing RTL clock gating. When you set this variable to **true**, **insert_clock_gating** does not split the load of the clock-gating cells (by duplicating the cells) to save area and power.

To determine the current value of this variable, use the **printvar power_hdlc_do_not_split_cg_cells** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

power_keep_license_after_power_commands

Affects the amount of time a Power Compiler license is checked out during a shell session.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable affects the amount of time a Power Compiler license is checked out during a shell session.

When set to **true**, a Power Compiler license that is checked out remains checked out throughout the shell session. When this variable is set to **false** (the default value), the Power Compiler license remains checked out only as long as a command is using it, and at the completion of the command, the license is released.

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

SEE ALSO

`report_power(2)`

power_lib2saif_rise_fall_pd

Specifies whether the **lib2saif** generates forward SAIF files with directives to generate rise and fall dependent path-dependent toggle counts.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When this variable is set to **true**, the **lib2saif** command generates forward SAIF files with directives to generate separate rise and fall values for non-state-dependent path-dependent toggle counts.

When this variable is set to **false**, directives to generate just the total of the rise and fall values for non-state-dependent path-dependent toggle counts are generated.

For more accurate power calculations, use separate rise and fall toggle counts; however, older simulators and simulation interfaces may not recognize such directives and will fail to read the library forward SAIF file. The Synopsys SAIF generation PLI utility provided with Power Compiler X-2005.09 supports directives for separate rise and fall values for path-dependent toggle rates, but older versions of the utility do not.

Note that this variable affects only directives for path-dependent toggle counts that are not state-dependent. Whether this variable is set to true or false, directives for separate rise and fall values are generated for state-dependent and both state and path-dependent toggle counts.

SEE ALSO

```
lib2saif(2)
```

power_min_internal_power_threshold

Specifies the minimum cell internal power value that can be used in power calculations.

TYPE

string

DEFAULT

""

GROUP

power_variables

DESCRIPTION

The value of this variable specifies the minimum threshold used for the cell internal power value. Internal power values are computed using the cell's switching activity and internal power characterization. If this variable has a numeric value and a cell's computed internal power is less than the variable's value, then the variable's value is used instead.

This variable has an effect only if it has a numeric value. The default value of this variable is "", which is non-numeric and so specifies that no minimum threshold is used on the cell internal power.

In general, this variable should not be used, since the internal power characterization specifies the correct internal power values.

SEE ALSO

`report_power(2)`

power_model_preference

Specifies the preference between the CCS power and the NLPM models in library cells that have power specified in both models.

TYPE

string

DEFAULT

nlpm

GROUP

power_variables

DESCRIPTION

A library can contain CCS power, NLPM, or both types of data within a cell definition. This variable specifies the power model preference if the library contains both NLPM and CCS power data.

Allowed values are as follows:

ccs instructs Power Compiler to use CCS power data in the library (if present) to calculate both static and dynamic power. If CCS power data is not found, Power Compiler uses NLPM data.

nlpm (the default) instructs Power Compiler to use NLPM data. If NLPM data is not found, Power Compiler uses CCS power data.

If neither CCS power nor NLPM data is found for a cell in the library, this cell is not characterized for power analysis.

SEE ALSO

`report_power(2)`

power_nets_for_upf_supply

Specifies a list of supply net names that identifies those nets as power (not ground) type supply nets in IC Compiler multivoltage UPF flow.

TYPE

String

DEFAULT

""

GROUP

mv

DESCRIPTION

This variable, when set to a list of valid UPF supply net names, identifies those nets as power (not ground) type supply nets in the IC Compiler multivoltage UPF flow. This can be useful in a UPF black-box flow, when some supply nets are used in subblocks but are not used at the top level. By identifying these nets as power supply nets, the IC Compiler tool has the information it needs to derive the newly created power supply nets at the top level, and the power type information is also honored by other multivoltage commands such as **check_mv_design** etc.

It is not necessary to use this variable to identify UPF supply nets that already have real usage in the design, for example, when they are used as the primary, isolation, or retention power supply, or are explicitly connected to power pins of leaf cells. In these cases, you can derive the power type for these supply nets and later implement them as real power nets by using the **derive_pg_connection -create_net** command.

By default, nothing is set for this variable, which means no supply nets are explicitly identified as power (not ground) nets.

The names listed in this variable must be actual supply net object names in the design. If any names do not match an existing supply net object name, those names are ignored, and you get a warning message. Furthermore, each listed name must be used only as a power

supply net, not a ground net, in the design. For example, if you use a listed net as a primary ground, the **check_mv_design** or **derive_pg_connection** will report a **MV-544** conflict error, and no PG net will be created.

SEE ALSO

```
ground_nets_for_upf_supply(3)
derive_pg_connection(2)
check_mv_design(2)
```

power_opto_extra_high_dynamic_power_effort

This variable is **obsolete**. Previous usage: Instructs the **compile** command to invoke more dynamic power optimization algorithms.

TYPE

Boolean

DEFAULT

false

GROUP

power

DESCRIPTION

This variable is obsolete. For dynamic power optimization, use the **set_dynamic_optimization true** in non-MCMM designs or **set_scenario_options -dynamic true** in MCMM designs. In both cases, you can also use **set power_low_power_placement true** to enable low power placement.

SEE ALSO

`set_scenario_options(2)`

power_preserve_rtl_hier_names

Preserves the hierarchy information of the RTL objects in the RTL design.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

When this variable is set to **true**, HDL Compiler preserves the hierarchy information of the RTL objects in the RTL design. This information is needed by the **rtl2saif** command, which generates RTL forward-annotation SAIF files. When this variable is **false** (the default value), **rtl2saif** cannot extract the correct synthesis invariant objects because the hierarchy information is not preserved.

To determine the current value of this variable, use the **printvar power_preserve_rtl_hier_names** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

power_rclock_inputs_use_clocks_fanout

Specifies whether clock network objects in an input port fanout are used to infer the input port's related clock.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

This variable is used during related clock inference to decide whether the inferred related clock on a design port is chosen to be the fastest clock whose clock network objects are in the port's transitive fanout. For example, when this variable is set to **true** (the default), if the transitive fanout of an input port contains a number of cells, the fastest clock on these flip-flop cells is chosen as the inferred related clock on the input port. When the variable is set to **false**, the input port will not have an inferred related clock.

For more information on the mechanism used to infer related clock information, see the **propagate_switching_activity** command man page.

SEE ALSO

```
propagate_switching_activity(2)
power_rclock_unrelated_use_fastest(3)
power_rclock_use_asynch_inputs(3)
```

power_rclock_unrelated_use_fastest

Specifies whether the fastest clock is set as the related clock of a design object when a related clock is not inferred by the related clock inference mechanism.

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

This variable is used during the last stage of related clock inference to decide whether or not the design objects that do not have an inferred related clock will be set as a related clock.

You can use the **set_switching_activity** command with the **-clock "*"** argument to specify that the tool will automatically infer the related clocks for any specified objects. If the related clock inference mechanism did not infer a related clock for a number of such objects, then when the value of the **power_rclock_unrelated_use_fastest** is **true**, the tool will set the fastest design clock as the objects' related clock. When the variable is set to **false**, such objects will not have a related clock.

For more information on the mechanism used to infer related clock information, see the **propagate_switching_activity** command man page.

SEE ALSO

```
propagate_switching_activity(2)
power_rclock_inputs_use_clocks_fanout(3)
power_rclock_use_asynch_inputs(3)
```

power_rclock_use_asynch_inputs

Specifies whether the inferred related clock on an asynchronous pin of a flip-flop is used to determine the inferred related clock on the cell's outputs.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

This variable is used during related clock inference to decide whether the inferred related clock on a flip-flop cell output considers the inferred related clocks on the cell's asynchronous inputs.

When this variable is set to **false**, the inferred related clock on a flip-flop cell output is the inferred related clock on the cell's clock pin. When this variable is set to **true**, the inferred related clock on a flip-flop cell output is the fastest inferred clock on the cell's clock pin and asynchronous input pins.

For more information on the mechanism used to infer related clock information, see the **propagate_switching_activity** command man page.

SEE ALSO

```
propagate_switching_activity(2)
power_rclock_inputs_use_clocks_fanout(3)
power_rclock_use_asynch_inputs(3)
```

power_remove_redundant_clock_gates

Specifies to the **compile -incremental** and **physopt -incremental commands to remove redundant Synopsys clock gating cells.**

TYPE

Boolean

DEFAULT

true

GROUP

power_variables

DESCRIPTION

This variable specifies whether to remove redundant Synopsys clock gating cells during incremental compile. A clock gate is considered redundant when it is always enabled. This is the case when the enable net is tied to logic one. When this variable is set to **true** (the default value), the redundant clock-gating cells are removed during incremental compile. To disable the automatic removal of redundant clock gating cells, set the value to **false** before issuing the **compile** or **physopt** commands.

To determine the current value of this variable, use the **printvar power_remove_redundant_clock_gates** command. For a list of power variables and their current values, use **print_variable_group power**.

SEE ALSO

power_same_switching_activity_on_connected_objects

Forces the tool to use the last user-annotated switching activity data on all connected tool objects.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

The **power_same_switching_activity_on_connected_objects** variable forces all switching activity data of all connected tool objects to use the latest user-annotated switching activity data. The user-annotated switching activity information can come from the **read_saif** command or the **set_switching_activity** command. Power reporting commands may produce different power results, since the switching activity data may be changed.

The **power_same_switching_activity_on_connected_objects** variable is off (false) by default.

SEE ALSO

```
read_saif(2)
set_switching_activity(2)
```

power_scale_internal_arc

Enables scaling of state probabilities of internal power arcs.

TYPE

Boolean

DEFAULT

false

GROUP

power_variables

DESCRIPTION

By default this variable is set to *false*, which disables scaling. If you set the Tcl variable to *true*, then the sum of the probabilities of the internal arc matches with the toggle rate of the pin even if there is no default arc for the pin in the library.

To find the current value of this variable, use the following command:

```
get_app_var power_scale_internal_arc
```

power_sdpd_message_tolerance

Specifies the tolerance value for issuing warning and information messages during fixing of user-annotated SDPD switching activity.

TYPE

float

DEFAULT

1e-05

GROUP

power_variables

DESCRIPTION

This variable specifies a tolerance value that is used when messages are reported during the SDPD fixing step. The SDPD fixing step is enabled by the **power_fix_sdpd_annotation** variable. See the **power_fix_sdpd_annotation** command man page for more information on this step.

Messages reported by the SDPD fixing step need to specify the tolerance criteria specified by this variable. For example, SDPD fixing scales state-dependent static probabilities so that they add up to 1.0.

A message is reported by the SDPD fixing step if the difference between the sum of the state-dependent static probabilities and 1.0 is not within the tolerance value. A tolerance value of 0.0 makes the SDPD fixing step report a message for every check and modification to the SDPD annotation; however, this will most likely report non-issues due to floating point errors. A high value of the **power_sdpd_message_tolerance** value filters out all but the most significant messages. Note that the verbosity of the SDPD fixing step is determined by the **power_fix_sdpd_annotation_verbose** variable. When **power_fix_sdpd_annotation_verbose** is set to **false**, most of the messages reported by the SDPD fixing step are replaced by a single message indicating that user-annotated SDPD switching activity is being corrected.

SEE ALSO

power_fix_sdpd_annotation(3)
power_fix_sdpd_annotation_verbose(3)

power_switch_attributes

Describes the attributes related to power switches.

DESCRIPTION

This man page described the attributes related to power switches. These attributes are defined only in UPF mode.

You can use the **get_attribute** command to determine value of an attribute, and use the **report_attribute** command to get a report \ of all the attributes on a specified power switch. To see all the power-switch attributes, use the **list_attribute -class power_switch -application** command.

Power Switch Attributes

cell_id

Specifies Milkyway design ID in which a power switch object is located.

This attribute is read-only.

name

Specifies name of a power switch object.

This attribute is read-only.

full_name

Specifies full name of a power switch object.

This attribute is read-only.

object_class

Specifies object class name of a power switch object, which is **power_switch**.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

This attribute is read-only.

within_block_abstraction

Specifies whether the power switch is part of the block abstraction.

The data type of **wthin_block_abstraction** is boolean.

This attribute is read-only and cannot be modified.

within_ilm

Specifies whether the power switch is part of an ILM.

The data type of **within_ilm** is boolean.

This attribute is read-only and cannot be modified.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
```

preroute_opt_verbose

Controls how verbose messages are displayed during DRC fixing, hold fixing, multiple-port-net fixing and tie-off optimization in the preroute stage. The debug can be used for debug purposes.

TYPE

Positive integer

DEFAULT

0

DESCRIPTION

This variable controls how verbose messages are displayed during DRC fixing, hold fixing, multiple-port-net fixing, tie-off and setup optimization in preroute stage. The messages can be used for debug purposes.

The default is 0.

When the variable is set to bitwise AND 2 (bit 2), which equals 1, the verbose messages are displayed during DRC fixing. For example, **set preroute_opt_verbose 2**.

When the variable is set to bitwise AND 4 (bit 3), which equals 1, the verbose messages are displayed during hold fixing. For example, **set preroute_opt_verbose 4**.

When the variable is set to bitwise AND 8 (bit 4), which equals 1, the verbose message are displayed during tie-off optimization. For example, **set preroute_opt_verbose 8**.

When the variable is set to bitwise AND 16 (bit 5), which equals 1, the verbose message are displayed during multiple-port-net fixing. For example, **set preroute_opt_verbose 16**.

When the variable is set to bitwise AND 32 (bit 6), which equals 1, the verbose message are displayed during setup fixing. For example, **set preroute_opt_verbose 32**.

When the variable is set to bitwise AND 128 (bit 8), which equals 1, the verbose message will be saved to file propt_verbose.log. Bit 8 only works with DRC and setup fixing. For example, set preroute_opt_verbose 0xa0 (output setup verbose message to file) set preroute_opt_verbose 0x82 (output DRC verbose message to file)

You can output verbose message to a file. The file name is by default set as propt_verbose.log and can not be changed.

You can use hexadecimal format as input , for example: set preroute_opt_verbose 0x2 (DRC verbose) set preroute_opt_verbose 0x4 (hold verbose) set preroute_opt_verbose 0x8 (tie-off verbose) set preroute_opt_verbose 0x10 (mpn verbose) set preroute_opt_verbose 0x20 (setup verbose)

A common error from usage is: set preroute_opt_verbose 8002

Above setting is not valid value infact and the result is unknown.

You can also set the variable to show messages in multiple areas. For example, set preroute_opt_verbose 24 (both tie-off and mpn verbose)

SEE ALSO

`routeopt_verbose(3)`

preserved_floating_nets

Configure the floating nets that need to be preserved and to be prevented from being removed during optimization.

TYPE

string

DEFAULT

""

GROUP

ICC infrastructure variable

DESCRIPTION

By default, the floating nets will be removed during optimization. User can set the variable **preserved_floating_nets** to avoid this. The variable should be set with a string concatenated with the net names separated by character " "(blank) or TAB of all the nets that needs to be preserved. Also one can specify asterisks (*), to preserve all the floating nets in the design.

This variable need to be set before the design being linked. It is recommended to set this variable just after **open_mw_cel** command.

EXAMPLES

Using preserved_floating_nets

The following example show how to use this tcl variable for preserving floating nets.

For example if there are 5 floating nets 'n1', 'n2', 'n3', 'A/n1', 'A/n2' in the design

```
prompt> open_mw_cel top
prompt> set preserved_floating_nets "*"
prompt> place_opt
```

```
Information: Floating net n1 is preserved. (MWDC-169)
Information: Floating net n2 is preserved. (MWDC-169)
Information: Floating net n3 is preserved. (MWDC-169)
Information: Floating net A/n1 is preserved. (MWDC-169)
Information: Floating net A/n2 is preserved. (MWDC-169)

prompt> open_mw_cel top
prompt> set preserved_floating_nets "n1 A/n1"
prompt> place_opt
Information: Floating net n1 is preserved. (MWDC-169)
Information: Floating net A/n1 is preserved. (MWDC-169)
```

SEE ALSO

psyn_onroute_disable_cap_drc

This variable turns on and off the max_cap fixing optimization in DRC fixing stage during route_opt, focal_opt and signoff_opt.

TYPE

Boolean

DEFAULT

""

GROUP

routeopt_variables

DESCRIPTION

If set to true, this variable disables fixing max capacitance violation during DRC fixing stage in route_opt, focal_opt and signoff_opt.

SEE ALSO

```
route_opt(2)
focal_opt(2)
signoff_opt(2)
```

psyn_onroute_disable_fanout_drc

This variable turns on and off the max_fanout optimization in DRC fixing stage during route_opt, focal_opt and signoff_opt.

TYPE

Boolean

DEFAULT

""

GROUP

routeopt_variables

DESCRIPTION

If set to true, this variable disables max_fanout optimization during DRC fixing stage in route _opt, focal_opt and signoff_opt.

SEE ALSO

```
route_opt(2)
focal_opt(2)
signoff_opt(2)
```

psyn_onroute_disable_hold_fix

This variable turns on and off the hold fixing optimization during route_opt.

TYPE

Boolean

DEFAULT

""

GROUP

routeopt_variables

DESCRIPTION

If set to true, this variable disables hold_fixing optimization during route_opt and hold fixing will not be performed during post-route optimization.

SEE ALSO

`route_opt(2)`

psyn_onroute_disable_netlength_drc

This variable turns on and off the max_net_length_fixing optimization in DRC fixing stage during route_opt, focal_opt and signoff_opt.

TYPE

Boolean

DEFAULT

""

GROUP

routeopt_variables

DESCRIPTION

If set to true, this variable disables fixing max_net_length violation during DRC fixing stage in route _opt, focal_opt and signoff_opt. Then, the tool will not add buffers to reduce net length even if the physical net length is more than constraints.

SEE ALSO

```
route_opt(2)
signoff_opt(2)
```

psyn_onroute_disable_trans_drc

This variable turns on and off the max transition fixing in DRC fixing stage during route_opt, focal_opt and signoff_opt.

TYPE

Boolean

DEFAULT

""

GROUP

routeopt_variables

DESCRIPTION

If set to true, this variable disables fixing max transition violation during DRC fixing stage in route _opt, focal_opt and signoff_opt.

SEE ALSO

```
route_opt(2)
focal_opt(2)
signoff_opt(2)
```

psyn_onroute_size_seqcell

Controls the sizing of sequential cells during postroute optimization.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable controls the sizing of sequential cells during postroute optimization.

When **false** (the default), sequential cell sizing is disabled during postroute optimization to minimize clock tree disturbance.

When set to **true**, this variable enables the sizing of sequential cells during postroute optimization.

SEE ALSO

```
route_opt(2)
focal_opt(2)
```

psyn_stress_map

Controls the generation of stress maps.

TYPE

Boolean

DEFAULT

false

GROUP

physopt

DESCRIPTION

Enables the generation of stress maps. The stress map shows how intensively the optimization (timing, DRC, power, and others) methods were applied to the current design. Set this variable to true to generate stress maps. The default value is false. It must be set to true before optimization to generate stress maps.

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

```
prompt> printvar psyn_stress_map
```

psynopt_density_limit

Sets the density limit for local region optimization. This limit prevents preroute optimization being performed on high density regions.

TYPE

float

DEFAULT

-1

GROUP

psynopt_variables

DESCRIPTION

This variable sets the density limit for checking a local region during preroute optimization stage.

When the tool tries to optimize a local region, it checks the region's density. If the density is greater or equal to the limit, then the density check fails, and the optimization process is NOT be performed. That means, the optimization tool does NOT perform buffer insertion or cell sizing in this high density region.

The variable affects the following optimization processes: power recovery, DRC fixing, timing fixing, hold fixing, and area recovery.

The variable affects the following optimization commands: psynopt, place_opt, clock_opt, preroute_focal_opt, place_opt_feasibility, and clock_opt_feasibility.

The default is -1. You should set the variable to a number less than 1.0. If you set the variable to a number larger than 1.0, the optimization might result in high local density.

psynopt_high_fanout_legality_limit

The maximum high fanout threshold for performing psynopt optimization tricks. This limit can be turned off by setting it to 0.

TYPE

Integer

DEFAULT

40

GROUP

psynopt_variables

DESCRIPTION

It sets the high fanout limit for optimization tricks to be performed. If the fanout number resulted from performing the current optimization trick is greater than this limit, the trick will have a good chance to be rejected. This feature aids in improvement of optimization performance by reducing high fanout nets. The benefit could be seen in runtime and general QoR. Setting this limit to 0 turns off this feature and optimization tricks will be performed regardless of the resulting fanout number. Please note that this variable does not impact the high fanout synthesis but controls the final fanout of nets. Also, it does not eliminate high fanout nets that existed before optimization stage.

SEE ALSO

```
place_opt(2)
clock_opt(2)
psynopt(2)
```

psynopt_tns_high_effort

Enables very high-effort optimization to fix total negative slack but increases the runtime.

TYPE

Boolean

DEFAULT

false

GROUP

psynopt_variables

DESCRIPTION

This variable causes preroute commands, such as **place_opt**, **clock_opt**, and **psynopt**, to use very high-effort optimization to fix total negative slack. This capability is not enabled by default because it increases the runtime.

EXAMPLES

The following example enables the **place_opt** command to invoke very high-effort optimization to fix total negative slack:

```
prompt> set_app_var psynopt_tns_high_effort true
prompt> psynopt
```

SEE ALSO

```
psynopt(2)
place_opt(2)
clock_opt(2)
```

query_objects_format

TYPE

string

DEFAULT

Legacy

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)
```

rail_indesign_shell_mode

Specifies whether PrimeRail version H-2013.06 or later is used to run rail analysis.

TYPE

string

DEFAULT

tcl

DESCRIPTION

Specifies whether or not PrimeRail version H-2013.06 or later is used for running In-Design rail analysis. If the value is set to tcl, PrimeRail version H-2013.06 or later is used. If the variable value is set to scheme, PrimeRail version G-2012.06 or older is used. The default value is tcl.

Avoid changing the value of this Tcl variable once set. Mixed usage between older and newer PrimeRail versions within the same rail analysis session within the IC Compiler environment might result in unpredictable results.

IC Compiler version H-2013.03-SP2 accompanies PrimeRail version H-2013.06 which supports static analysis only. You need to set the **rail_indesign_shell_mode** variable to scheme for invoking the PrimeRail version G-2012.06 executable if you want to perform dynamic analysis. You do not need to specify the **rail_indesign_shell_mode** variable if you want to perform only static analysis.

When you use PrimeRail versions G-2012.06 or older, set the value to scheme before invoking any of the rail analysis commands, including **set_rail_options**, **report_rail_options**, and **analyze_rail**. The value of this Tcl variable affects the behavior of all three commands. The legacy man pages for these commands are analyze_rail_scheme_mode.2, set_rail_options_scheme_mode.2, and report_rail_options_scheme_mode.2.

Use the following command to determine the current value of the variable:

```
prompt> printvar rail_indesign_shell_mode
```

SEE ALSO

```
set_rail_options(2)
report_rail_options(2)
analyze_rail(2)
```

rc_degrade_min_slew_when_rd_less_than_rnet

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

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables the use of slew degradation during the RCCALC-009 condition. When set to **false**, (the default), slew degradation through RC networks is not used in minimum analysis mode during the RCCALC-009 condition.

The "RCCALC-009 condition" means a condition in which timing analysis 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 a particular drive-strength threshold, timing analysis adjusts the drive resistance using an empirical formula to improve accuracy, and issues the RCCALC-009 message. If this improved accuracy is not sufficient, timing analysis provides extra pessimism by not using slew degradation in minimum analysis mode; however, unnecessary minimum delay violations could occur as a side effect. You can keep slew degradation on in minimum analysis mode after you have qualified the RCCALC-009 methodology for your accuracy requirements, by setting this variable to **true**.

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

SEE ALSO

rc_driver_model_mode

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

TYPE

string

DEFAULT

advanced

GROUP

timing_variables

DESCRIPTION

This variable specifies whether the timing engine uses a basic or advanced driver model for RC delay calculation. The **basic** model is derived from a simpler delay and slew library method, whereas the **advanced** model is derived from an advanced driver model that is part of the Synopsys Composite Current Source (CCS) model.

When the variable is set to **basic**, RC delay calculation always uses driver models derived from the conventional delay and slew model present in the design library. When it is set to **advanced** (the default), RC delay calculation uses the advanced driver model, if data for the model is present. In that case, when you use the **report_delay_calculation** command to report the cell arc, it displays the message "Advanced driver-modeling used".

You do not need to set the **rc_driver_model_mode** variable to **advanced** to enable the CCS driver model. The CCS model is used automatically if the CCS libraries exist. However, you can set the **rc_driver_model_mode** variable to **basic** to disable the advanced CCS driver model.

When the **rc_driver_model_mode** variable is set to **basic** and the **rc_receiver_model_mode** variable is set to **advanced**, the timing engine uses the advanced voltage-dependent capacitance models to derive an equivalent single capacitance model that depends only on the rise, fall, minimum, or maximum arc condition. These equivalent capacitance values are used in the analysis instead of the pin capacitance values from the library. For details, see the **rc_receiver_model_mode** variable man page.

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

SEE ALSO

`report_delay_calculation(2)`
`rc_receiver_model_mode(3)`

rc_input_threshold_pct_fall

Specifies the threshold voltage that defines the startpoint of the falling cell or net delay calculation.

TYPE

float

DEFAULT

50

GROUP

timing_variables

DESCRIPTION

This variable specifies the threshold voltage that defines the startpoint of the falling cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 through 100.0 inclusive.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trip-point values specified in the library, so it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trip-point specifications.

Table 1

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

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

To determine the current value of this variable, use the **printvar rc_input_threshold_pct_fall** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

EXAMPLES

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

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

SEE ALSO

```
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_input_threshold_pct_rise

Specifies the threshold voltage that defines the startpoint of the rising cell or net delay calculation.

TYPE

float

DEFAULT

50

GROUP

timing_variables

DESCRIPTION

This variable specifies the threshold voltage that defines the startpoint of the rising cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 through 100.0 inclusive.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trip-point values specified in the library, so normally it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trip-point specifications.

Table 1

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

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

To determine the current value of this variable, use the **printvar rc_input_threshold_pct_rise** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

EXAMPLES

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

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

SEE ALSO

```
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_noise_model_mode

Enables the use of CCS noise, if available in the design library, when set to **advanced**.

TYPE

string

DEFAULT

basic

GROUP

signal integrity

DESCRIPTION

This variable enables CCS noise in static noise analysis and optimization, when set to **advanced**.

When set to **basic** (the default), CCS noise information is not used even if it exists in the library. However, if the library has NLDM noise information, it is still used.

To determine the current value of this variable, use the **get_app_var rc_noise_model_mode** command.

SEE ALSO

```
report_noise(2)
rc_driver_model_mode(3)
rc_receiver_model_mode(3)
```

rc_output_threshold_pct_fall

Specifies the threshold voltage that defines the startpoint of the falling cell or net delay calculation.

TYPE

float

DEFAULT

50

GROUP

timing_variables

DESCRIPTION

This variable specifies the threshold voltage that defines the startpoint of the falling cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 through 100.0 inclusive.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trip-point values specified in the library, so it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trip-point specifications.

Table 1

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

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

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

For a list of all timing variables and their current values, type **print_variable_group timing**.

EXAMPLES

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

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

SEE ALSO

```
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_output_threshold_pct_rise

Specifies the threshold voltage that defines the startpoint of the rising cell or net delay calculation.

TYPE

float

DEFAULT

50

GROUP

timing_variables

DESCRIPTION

This variable specifies the threshold voltage that defines the startpoint of the rising cell or net delay calculation. The value is a percent of the voltage source. Allowed values are 0.0 through 100.0 inclusive.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trip-point values specified in the library, so it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trip-point specifications.

Table 1

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

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

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

For a list of all timing variables and their current values, use **print_variable_group timing**.

EXAMPLES

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

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

SEE ALSO

```
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_receiver_model_mode

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

TYPE

string

DEFAULT

advanced

GROUP

timing_variables

DESCRIPTION

This variable specifies whether the timing engine uses a basic or advanced receiver model for RC delay calculation. The **basic** model is a single capacitance model that depends only on the rise, fall, minimum, or maximum arc condition. The **advanced** model is a voltage-dependent capacitance model that also depends on input slew and output capacitance.

One advantage of the advanced model is improved accuracy for both delay and slew calculation. Another advantage is that it considers nonlinear effects such as the Miller effect. The advanced receiver model is part of the Synopsys Composite Current Source (CCS) model.

When this variable is set to **advanced** (the default), RC delay calculation uses the advanced receiver model if data for that model is present and if the network driver uses the advanced driver model. In that case, a report generated by the **report_delay_calculation** command for a network arc displays the message "Advanced receiver-modeling used".

You do not need to set the **rc_receiver_model_mode** variable to **advanced** to enable the CCS receiver model. The CCS model is used automatically if the CCS libraries exist. However, you can set the **rc_receiver_model_mode** variable to **basic** to disable the advanced CCS receiver model.

When the **rc_receiver_model_mode** variable is set to **advanced**, and the network is not driven by the advanced driver model (such as when the **rc_driver_model_mode** variable is

set to basic or a lumped load is used), the timing engine uses the advanced voltage-dependent capacitance models to derive an equivalent single capacitance model that depends only on the rise, fall, minimum, or maximum arc condition. These equivalent capacitance values are used in analysis instead of the pin capacitance values from the library. The **report_delay_calculation** command used on a network arc does not show the "Advanced receiver-modeling used" message for these calculations because only an equivalent single capacitance is used.

When the **rc_receiver_model_mode** variable is set to **basic**, RC delay calculation uses the pin capacitance values specified in the design libraries.

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

SEE ALSO

```
report_delay_calculation(2)
rc_driver_model_mode(3)
```

rc_slew_derate_from_library

Specifies the derating needed for the transition times in the Synopsys library to match the transition times between the characterization trip points.

TYPE

float

DEFAULT

1

GROUP

timing_variables

DESCRIPTION

This variable specifies a floating-point number between 0.0 and 1.0 that indicates the derating needed for the transition times in the Synopsys library to match the transition times between the characterization trip points. The default is 1.0, which means that the transition times in the Synopsys library are used without change.

The value this variable specifies is overridden by any library-specified slew-derating values, so it should not be necessary to set the variable. The value specified applies only to libraries that do not contain slew-derating specifications.

A slew-derating value of 1.0 should be used if the transition times specified in the library represent the exact transition times between the characterization trip points, which is usually the case. Use a slew-derating value of less than 1.0 for libraries where the transition times have been extrapolated to the rail voltages. For example, if the transition times are characterized as between 30 percent and 70 percent and then extrapolated to the rails, the slew-derating value is 0.4 = (70 - 30) / 100.

To determine the current value of this variable, use the **printvar rc_slew_derate_from_library** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

EXAMPLES

The following example specifies that cell delays from the Synopsys library have been computed from 50 percent of the input transition to 50 percent of the output transition. The example also specifies that transition times in the Synopsys library were computed by measuring the delay from 30 percent to 70 percent of the voltage source and then multiplying the measured transition times by 2.5 = (100-0)/(70-30) to extrapolate to 0-100 percent of the rail voltages.

```
prompt> set rc_slew_derate_from_library 0.4
prompt> set rc_slew_lower_threshold_pct_fall 30
prompt> set rc_slew_lower_threshold_pct_rise 30
prompt> set rc_slew_upper_threshold_pct_fall 70
prompt> set rc_slew_upper_threshold_pct_rise 70
prompt> set rc_input_threshold_pct_rise      50
prompt> set rc_input_threshold_pct_fall      50
prompt> set rc_output_threshold_pct_rise     55
prompt> set rc_output_threshold_pct_fall     55
```

SEE ALSO

```
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_slew_lower_threshold_pct_fall

Specifies the threshold voltage that defines the endpoint of the falling slew calculation.

TYPE

float

DEFAULT

20

GROUP

timing_variables

DESCRIPTION

This variable specifies the threshold voltage that defines the endpoint of the falling slew calculation. The value is a percent of the voltage source. Allowed values are 0.0 through 100.0 inclusive.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trip-point values specified in the library, so it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trip-point specifications.

Table 1

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

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

To determine the current value of this variable, use the **printvar rc_slew_lower_threshold_pct_fall** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

EXAMPLES

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

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

SEE ALSO

```
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_slew_lower_threshold_pct_rise

Specifies the threshold voltage that defines the startpoint of the rising slew calculation.

TYPE

float

DEFAULT

20

GROUP

timing_variables

DESCRIPTION

This variable specifies the threshold voltage that defines the startpoint of the rising slew calculation. The value is a percent of the voltage source. Allowed values are 0.0 through 100.0 inclusive.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trip-point values specified in the library, so it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trip-point specifications.

Table 1

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

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

To determine the current value of this variable, use the **printvar rc_slew_lower_threshold_pct_rise** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

EXAMPLES

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

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

SEE ALSO

```
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_slew_upper_threshold_pct_fall

Specifies the threshold voltage that defines the startpoint of the falling slew calculation.

TYPE

float

DEFAULT

80

GROUP

timing_variables

DESCRIPTION

This variable specifies the threshold voltage that defines the startpoint of the falling slew calculation. The value is a percent of the voltage source. Allowed values are 0.0 through 100.0 inclusive.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trip-point values specified in the library, so it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trip-point specifications.

Table 1

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

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

To determine the current value of this variable, use the **printvar rc_slew_upper_threshold_pct_fall** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

EXAMPLES

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

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

SEE ALSO

```
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

rc_slew_upper_threshold_pct_rise

Specifies the threshold voltage that defines the endpoint of the rising slew calculation.

TYPE

float

DEFAULT

80

GROUP

timing_variables

DESCRIPTION

This variable specifies the threshold voltage that defines the endpoint of the rising slew calculation. The value is a percent of the voltage source. Allowed values are 0.0 through 100.0 inclusive.

This variable is one of 8 variables, listed in Table 1, that affect delay and transition time computations for detailed RC networks. These variables interpret the cell delays and transition times from the Synopsys library. The values specified by these variables are overridden by trip-point values specified in the library, so it should not be necessary to set the variables. The values specified are only applied to libraries that do not contain trip-point specifications.

Table 1

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

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

To determine the current value of this variable, use the **printvar rc_slew_upper_threshold_pct_rise** command. For a list of all timing variables and their current values, use **print_variable_group timing**.

EXAMPLES

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

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

SEE ALSO

```
rc_input_threshold_pct_fall(3)
rc_input_threshold_pct_rise(3)
rc_output_threshold_pct_fall(3)
rc_output_threshold_pct_rise(3)
rc_slew_lower_threshold_pct_fall(3)
rc_slew_lower_threshold_pct_rise(3)
rc_slew_upper_threshold_pct_fall(3)
rc_slew_upper_threshold_pct_rise(3)
```

read_db_lib_warnings

Indicates that warnings are to be printed while a technology .db library is being read in with the **read** command. When false (the default), no warnings are given.

TYPE

Boolean

DEFAULT

false

GROUP

io_variables

DESCRIPTION

Indicates that warnings are to be printed while a technology .db library is being read in with the **read** command. When false (the default), no warnings are given.

To determine the current value of this variable, type **printvar read_db_lib_warnings**. For a list of all **io** variables and their current values, type **print_variable_group io**.

SEE ALSO

```
read(2)
io_variables(3)
```

read_only_attributes

Contains informational attributes, which the user cannot set.

DESCRIPTION

Contains informational attributes. A "read-only" attribute cannot be set by the user.

To determine the value of an attribute, use the **get_attribute** command.

For information on all attributes, refer to the **attributes** manual page.

Read-only Attributes

design_type

Indicates the current state of the design and has the value *fsm* (finite state machine), *pla* (programmable logic array), *equation* (Boolean logic), or *netlist* (gates). This attribute cannot be set by the user.

is_black_box

true if the reference is not yet linked to a design. This attribute cannot be set by the user.

is_combinational

true if all cells of a design and all designs in its hierarchy are combinational. A cell is combinational if it is non-sequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command will report such a cell as not a black-box. This attribute cannot be set by the user.

is_dw_subblock

true if the object (a cell, a reference, or a design) is a DW subblock that was automatically elaborated. This attribute cannot be set by the user.

Note: DW subblocks that are manually elaborated will not have this attribute.

is_hierarchical

true if the design contains leaf cells or other levels of hierarchy. This attribute is read-only and cannot be set by the user.

is_mapped

true if all the non-hierarchical cells of a design are mapped to cells in a technology library. This attribute cannot be set by the user.

is_sequential

true if any cells of a design or designs in its hierarchy are sequential. A cell is sequential if it is not combinational. This attribute cannot be set by the user.

is_test_circuitry

Set by **insert_dft** on the scan cells and nets added to a design during the addition of test circuitry. This attribute cannot be set by the user.

is_synlib_module

true if the object (a cell, a reference, or a design) refers to an unmapped module reference or if the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute cannot be set by the user.

Note: synlib modules that are manually elaborated will not have this attribute.

is_synlib_operator

true if the object (a cell or a reference) is a synthetic library operator reference. This attribute cannot be set by the user.

is_unmapped

true if any of the cells are not linked to a design or mapped to a technology library. This attribute cannot be set by the user.

pin_direction

Direction of a pin. Value can be *in*, *out*, *inout*, or *unknown*. This attribute cannot be set by the user.

port_direction

Direction of a port. Value can be *in*, *out*, *inout*, or *unknown*. This attribute cannot be set by the user.

ref_name

The reference name of a cell. This attribute cannot be set by the user.

SEE ALSO

```
get_attribute(2)
attributes(3)
```

read_translate_msff

Automatically translates master-slave flip-flops (specified with the clocked_on_also syntax) to master-slave latches.

TYPE

Boolean

DEFAULT

true

GROUP

io_variables

DESCRIPTION

This variable automatically translates master-slave flip-flops (specified with the clocked_on_also syntax) to master-slave latches, when set to **true** (the default). When set to **false**, both master and slave remain flip-flops.

This variable is used when running the **read_file** command, while a technology .db library is being read in by the shell, and when running the **read_lib** command while a technology library is being read in by Library Compiler. The technology .db library is affected only if the program reports that the .db library is being updated and asks you to save the results. Library Compiler always follows this variable during processing.

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

SEE ALSO

```
read_file(2)
read_lib(2)
```

reference_attributes

Contains attributes that can be placed on a reference.

DESCRIPTION

Contains attributes that can be placed on a reference.

Several commands exist that can be used to set attributes; however, most attributes can be set by using the **set_attribute** command. If the attribute definition specifies a **set** command, use it to set the attribute; otherwise, use **set_attribute**. If an attribute is read-only, the you cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, see the manual page of the appropriate **set** command. For information on all attributes, refer to the **attributes** manual page.

Reference Attributes

dont_touch

Specifies that designs linked to a reference with this attribute are excluded from optimization. Valid values are true (the default) or false. Designs linked to a reference by using the **dont_touch** attribute set to true are not modified or replaced during compile. Set this by using the **set_dont_touch** attribute.

is_black_box

This is set to true if the reference is not yet linked to a design. This attribute is read-only and you cannot set it.

is_combinational

This is set to true if all the cells of the referenced design are combinational. A cell is combinational if it is nonsequential or non-tristate and all of its outputs compute a combinational logic function. The **report_lib** command reports such a cell as not a black-box. This attribute is read-only and you cannot set it.

is_dw_subblock

This is set to true if the object (a cell, a reference, or a design) is a DesignWare subblock that was automatically elaborated. This attribute is read-only and you cannot set it.

Note: DesignWare subblocks that are manually elaborated do not have this attribute.

is_hierarchical

This is set to true if the design contains leaf cells or other levels of hierarchy. This attribute is read-only and you cannot set it.

is_mapped

This is set to true if the reference is linked to a design, and all the non-hierarchical cells of the referenced design are mapped to cells in a technology library. This attribute is read-only and you cannot set it.

is_sequential

This is set to true if all the cells of the referenced design are sequential. A cell is sequential if it is not combinational (if any of its outputs depend on previous inputs). This attribute is read-only and you cannot set it.

is_synlib_module

This is set to true if the object (a cell, a reference, or a design) refers to an unmapped module reference, or the object is (or refers to) a design that was automatically elaborated from a synlib module or a synlib operator. This attribute is read-only and you cannot set it.

Note: synlib modules that are manually elaborated do not have this attribute.

is_synlib_operator

This is set to true if the object (a cell or a reference) is a synthetic library operator reference. This attribute is read-only and you cannot set it.

is_unmapped

This is set to true if any of the non-hierarchical cells of the referenced design are not mapped to cells in a technology library, or the reference is not yet linked to a design. This attribute is read-only and you cannot set it.

scan

When *true*, specifies that cells of the referenced design are always replaced by equivalent scan cells. When false, cells are not replaced.

scan_chain

Includes the specified cells of the referenced design in the scan-chain whose index is the value of this attribute.

ungroup

Specifies that all designs linked to a reference with this attribute are ungrouped (levels of hierarchy represented by these design cells are removed) during compile. Set by using the **set_ungroup** command.

SEE ALSO

```
get_attribute(2)
remove_attribute(2)
set_attribute(2)
attributes(3)
```

register_duplicate

Specifies that the **compile** command is to invoke register duplication to reduce the number of fanouts for each register.

TYPE

Boolean

DEFAULT

false

GROUP

compile_variables

DESCRIPTION

The **register_duplicate** variable when set to **true**(default is false), duplicates the high fanout registers to reduce the number of fanouts for each register. The max fanout that each register should drive is specified by setting set_max_fanout. The register replication occurs provided that you specified max fanout constraint using set_max_fanout.

Fanouts for registers that exceed a certain limit can cause an adverse affect on the circuit performance. The fanout is by default controlled only by buffer insertion. Some FPGA architectures lack buffers in the routing architecture that might cause the fixed fanout limit to exceed and cause an error in the back-end tool. To address this problem, the **register_duplicate** variable provides a method to duplicate registers in an optimized manner.

EXAMPLES

The following is an example of the use of **register_duplicate**:

```
prompt> set_max_fanout 25 top
prompt> register_duplicate = true
prompt> set_attribute xfpga_virtex2-5 \
```

```
        -type float default_fanout_load 100
```

The **compile** command indicates the register duplication status as ":-". After the report is generated, the following message appears:

```
Information: Duplicating register r1_reg with fanout load of 50.00
(REGDUP-3)

Optimization Complete
```

SEE ALSO

```
set_max_fanout(2)
```

register_replication_naming_style

Specifies the style to use in naming the replicated register with the **set_register_replication** command.

TYPE

string

DEFAULT

%s_rep%d

DESCRIPTION

This variable specifies the naming style of the replicated register created by the **set_register_replication** command. It is a string that has exactly one %s and one %d. The %s is replaced by the base name of the register. The %d is the i-th copy of the replicated register.

For example, if the variable is %s_rep_%d, then the first copy of the u0_reg register is u0_reg_rep_1. The fifth copy of the u6 register is u6_rep_5.

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

SEE ALSO

remove_route_guide_on_fat_pin

Removes the system metal blockage layers or route guides from the thin part of fat pins.

TYPE

Boolean

DEFAULT

false

GROUP

extract_blockage_pin_via_variables

DESCRIPTION

This variable removes the system metal blockage layers or route guides from the thin part of fat pins. Note that no via regions are generated on the thin part of fat pins after running the **extract_blockage_pin_via** command.

SEE ALSO

`extract_blockage_pin_via(2)`

reoptimize_design_changed_list_file_name

Creates a file in which to store the list of cells that changed and cells and nets that were added during post-layout or in-place optimization.

TYPE

string

DEFAULT

""

GROUP

compile_variables
links_to_layout_variables

DESCRIPTION

Creates a file in which to store the list of cells that changed and cells and nets that were added during post-layout or in-place optimization.

When the **reoptimize_design_changed_list_file_name** variable is set to a particular file name, the **reoptimize_design -in_place** and **reoptimize_design -post_layout_opto** commands output to the file a list of design modifications and additions. The commands use the file to store the list of those cells that changed and those cells and nets that were added. By default, this variable is set to an empty string, and no such file is created during post-layout or in-place optimization.

For backwards compatibility, the **compile -in_place** command also sends the list of changes to this file.

If the specified file already exists, the new changes and additions are appended to the existing file.

During in-place optimization, cells are not normally added. However, if the **compile_ok_to_buffer_during_inplace_opt** flag is asserted, in-place optimization might create new cells. If it does, those cells are included in the specified change-list file. New nets are also included in the file.

During post-layout optimization, net changes are not tracked. Consequently, a cell is considered changed if its library cell has changed. This is different from in-place optimization, where a cell is considered changed if any of its connected nets are changed.

To determine the current value of this variable, use **printvar reoptimize_design_changed_list_file_name**. For a list of all **compile** variables and their current values, use the **print_variable_group compile** command. For a list of all **links_to_layout** variables and their current values, use the **print_variable_group links_to_layout** command.

SEE ALSO

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 set this variable to be true, the advanced CCS receiver model is used to report receiver pin capacitance by the **report_net**, **report_constraint**, **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_constraint(2)
report_delay_calculation(2)
```

report_default_significant_digits

Sets the default number of significant digits for many reports.

TYPE

integer

DEFAULT

-1

DESCRIPTION

The **report_default_significant_digits** variable sets the default number of significant digits for many reports. Allowed values are 0-13; the default is -1. A value of -1 indicates that a command-specific default precision value is used for reporting. Some report commands, such as **report_timing** and **report_cell**, have a **-significant_digits** option, which overrides the value of this variable.

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

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

Once set, the value of the variable is used by the subsequent reporting commands if a command-specific **-significant_digits** option is not used. The value of the variable can be reset by setting the value of the variable to -1. This causes the command-specific default to be used as the precision for reporting if the **-significant_digits** command option is not used.

For example, assume the command-specific default precision of the **report_cell** command is 6. If the **report_default_significant_digits** variable is set to 5 and the **report_cell** command is run thereafter, a precision of 5 is used in the **report_cell** report. If the **report_cell -significant_digits 3** command is run after the above variable setting, a precision of 3 is used for the **report_cell** report. To view the **report_cell** report with the command default precision of 6, reset the value of the **report_default_significant_digits** variable by setting it to -1.

If an invalid value is set (not in the range 0-13 and not -1), an error is issued and the previous valid value is restored. The following example illustrates the usage of the variable:

```
prompt> printvar report_default_significant_digits
```

```
report_default_significant_digits = "-1"

prompt> set_app_var report_default_significant_digits 4
4

prompt> printvar report_default_significant_digits
report_default_significant_digits = "4"

prompt> set_app_var report_default_significant_digits -44
Error: can't set report_default_significant_digits: must be in range 0 to
13
        Use error_info for more info. (CMD-013)
4

prompt> printvar report_default_significant_digits
report_default_significant_digits = "4"
```

SEE ALSO

report_cell(2)
report_clock_gating_check(2)
report_constraint(2)
report_net(2)
report_qor(2)
report_timing(2)
write_sdf(2)

report_timing_use_accurate_delay_symbol

Specifies whether to use accurate delay symbols in **report_timing** reports.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

This variable specifies whether to use accurate delay symbols in the "Incr" column of reports generated by **report_timing**. When the variable is set to true (the default), the following symbols are used:

```
&: Delay information is timing-annotated using Elmore delay calculation
c: Delay information is from Arnoldi calculation with accurate CCS
@: Delay information is from Arnoldi calculation
a: Delay information is from postroute AWE calculation
w: Delay information is from preroute AWE calculation
```

When the variable is set to false, the "c", "@" and "a" symbols are not used; preroute delay increments are marked with "*" and postroute delay increments are marked with "&". The asterisk symbol (*) means back-annotation using preroute Elmore extraction or SDF.

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

SEE ALSO

`report_timing(2)`

rom_auto_inferring

Infers ROM from the RTL description.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true**, the tool attempts to infer ROM from the RTL description. When set to **false**, no attempt is made to infer ROM from the RTL description.

SEE ALSO

route_doubleViaDriven

Instructs the classic router to consider space allocation for double via insertion.

TYPE

Integer. 0 ("no") or 1 ("yes").

DEFAULT

0

DESCRIPTION

Insertion of double vias is done by the classic router **insert_redundant_vias** command. This command is applied after routing is completed.

By default, during routing, the classic router does not reserve space for double via insertion.

When the **route_doubleViaDriven** variable is set to 1, the classic router reserves space to enable easy insertion of double vias.

Note that reserving space for double vias might reduce routabilty.

EXAMPLE

To force the classic router to reserve space for double vias, enter the following command:

```
prompt> set_app_var route_doubleViaDriven 1
```

SEE ALSO

```
insert_redundant_vias(2)
droute_highEffortViaDoubling(3)
route_detail(2)
route_search_repair(2)
route_group(2)
route_area(2)
route_eco(2)
route_auto(2)
```

```
route_opt(2)
```

route_guide_attributes

Contains attributes related to route guide.

DESCRIPTION

Contains attributes related to route guide.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class route_guide -application**, the definition of attributes can be listed.

Route Guide Attributes

affects

Specifies the affects of a route guide, which is route.

The data type of **affects** is string.

This attribute is read-only.

area

Specifies area of a route guide.

The data type of **area** is float.

This attribute is read-only.

bbox

Specifies the bounding-box of a route guide. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

cell_id

Specifies Milkyway design ID in which a route guide object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

double_pattern_mask_constraint

Specifies the coloring information of the route_guide. This information is needed in the double patterning flow.

The data type of **double_pattern_mask_constraint** is string.

The valid values can be: any_mask, mask1_soft, mask1_hard, mask2_soft, mask2_hard and same_mask.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

horizontal_track_utilization

Specifies the horizontal track utilization for the route guide.

The data type of **horizontal_track_utilization** is integer.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

layer

Specifies layer name on which a route guide is.

The data type of **layer** is string.

This attribute is read-only.

layer_number

Specifies layer number on which a route guide is.

The data type of **layer_number** is integer.

This attribute is read-only.

multiple_pattern_mask_constraint

Specifies the coloring information of the route_guide. This information is needed in the multiple patterning flow.

The data type of **multiple_pattern_mask_constraint** is string.

The valid values can be: any_mask, mask1_soft, mask1_hard, mask2_soft, mask2_hard, mask3_soft, mask3_hard and same_mask.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name

Specifies name of a route guide object.

The data type of **name** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

no_preroute_layers

Specifies the layers that cannot contain preroutes.

The data type of **no_preroute_layers** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

no_signal_layers

Specifies the layers that cannot contain signals.

The data type of **no_signal_layers** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

object_class

Specifies object class name of a route guide, which is **route_guide**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

The data type of **object_id** is integer.

This attribute is read-only.

object_type

Specifies geometry type of a route guide, which can be RECTANGLE or POLYGON.

The data type of **object_type** is string.

This attribute is read-only.

points

Specifies point list of a route guide's boundary.

The data type of **points** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

preferred_direction_only_layers

Specifies the layers that cannot make nonPreferredDirection wires.

The data type of **preferred_direction_only_layers** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

repair_as_single_sbox

Specifies whether this route guide should be repaired as a single sbox when there is a difficult violation on a prerouted wire or inside a large macro.

The data type of **repair_as_single_sbox** is boolean.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

route_guide_group_id

Route guides with the same route guide group id belong to the same route guide group.

switch_preferred_direction

Specifies whether to switch the preferred direction for the route guide.

The data type of **switch_preferred_direction** is boolean.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

switch_preferred_direction_layers

Specifies the layers on which single layer routing should be encouraged.

The data type of **switch_preferred_direction_layers** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

track_utilization_layers

Specifies the layers on which horizontal_track_utilization or vertical_track_utilization had been set.

The data type of **track_utilization_layers** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

type

Specifies the affects of a route guide, which is route.

The data type of **type** is string.

This attribute is read-only.

vertical_track_utilization

Specifies the vertical track utilization for the route guide.

The data type of **vertical_track_utilization** is integer.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

zero_min_spacing

Specifies whether zero minimum spacing is allowed for the route guide.

The data type of **zero_min_spacing** is boolean.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

route_layerExtraCostByRC

DESCRIPTION

The **route_layerExtraCostByRC** variable specifies extra layer cost. It will control the choice of layers for detailed router to reduce RC. If the value is set to 0, user specified extra cost for all layers will be used. If the value is set to 1, extra cost of all poly and metal layers previously defined as 0 will be re-computed based on RC. If the value is set to 2, extra cost of all poly, metal layers, and vias previously defined as 0 will be re-computed based on RC.

SYNTAX

```
     set route_layerExtraCostByRC value

The valid values of this variable range between 0 and 2.
The default is 0.
```

EXAMPLE

To compute extra layer cost, enter

set route_layerExtraCostByRC 1

route_noVoltAreaFeedThru

Controls whether the classic router can route through voltage areas.

TYPE

Integer. 0 ("no") or 1 ("yes").

DEFAULT

1

DESCRIPTION

A connection through a voltage area is called a "voltage area feedthrough." By default, the classic router does not route connections through foreign voltage areas. You can change this behavior by setting this variable to 0.

EXAMPLE

To enable classic router connections to go through voltage areas, enter the following command:

```
prompt> set_app_var route_noVoltAreaFeedThru 0
```

SEE ALSO

```
route_detail(2)
route_search_repair(2)
route_group(2)
route_area(2)
route_eco(2)
route_auto(2)
route_opt(2)
```

routeopt_allow_min_buffer_with_size_only

Allows the post-route hold fixing to add buffers even in size-only mode

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

By the definition of size-only mode, the optimization stage is not supposed to add any buffers. If this variable is set to true, even if the post-route optimization is in size-only mode, post-route hold fix adds buffers in order to fix hold time violation. Thus, buffering is allowed only for hold fixing.

SEE ALSO

`route_opt(2)`

routeopt_checkpoint

Saves an intermediate design at intervals during the **route_opt** command.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

When this variable is set to true, the **route_opt** command saves intermediate designs at each major stage. For multiple runs of the **route_opt** command, each intermediate design is overwritten. The checkpoint design names are :

design_name_RT for after the initial routing stage
design_name_ORPRT for after the optimize wires and vias stage
design_name_OR_number for after each optimization stage
design_name_ECORT_number for after each ECO routing stage
design_name_RCRED for after the crosstalk reduction stage
design_name_POR for after the power optimization stage
design_name_PECORT for after the power optimization ECO routing stage
design_name_HOLD_PRE for after the preroute-based hold optimization stage
design_name_HOLD for after the global-route-based hold optimization stage
design_name_INCOR for after the optimization stage during **route_opt -incremental**
design_name_INCECORT for after the ECO routing stage during **route_opt -incremental**

SEE ALSO

route_opt(2)

routeopt_density_limit

Specifies the maximum cell density threshold for optimizations performed by the **route_opt** or **focal_opt** command in a window around the location of a cell or pin.

TYPE

float

DEFAULT

0.95

GROUP

routeopt_variables

DESCRIPTION

This variable sets the density limit for checking a window around a location where an onroute optimization is performed by the **route_opt** or **focal_opt** command. The window size is set internally and is based on the size of a default cell. If the density in the window area is greater than or equal to this limit, the density check fails and the optimization is not performed in that location.

The check is performed for dynamic buffering, maximum length buffering, minimum buffering, move, phase, and sizing optimizations. For sizing optimizations, it is checked only if the candidate library cell is larger than the candidate cell.

SEE ALSO

```
route_opt(2)
focal_opt(2)
```

routeopt_disable_cpulimit

Controls whether the internally defined threshold for the number of changes in one optimization is disabled.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

In the **route_opt** command, there is an upper limit for the number of changes in one optimization stage to help improve convergence.

When this variable is **false** (the default), the **route_opt** command stops optimization after it reaches this limit and moves on to the next stage.

When this variable is set to **true**, the **route_opt** command ignores this limit.

SEE ALSO

`route_opt(2)`

routeopt_drc_over_timing

Specifies the cost priority between the maximum design rule cost and the maximum delay cost during DRC fixing by the **route_opt** command.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable specifies the cost priority between the maximum design rule cost and the maximum delay cost during DRC fixing by the **route_opt** command.

When **false** (the default), the maximum delay cost has a higher priority than the maximum design rule cost.

When set to **true**, the maximum design rule cost has a higher priority then the maximum delay cost.

This variable works only with the **route_opt -only_design_rule** option.

SEE ALSO

`route_opt(2)`

routeopt_enable_aggressive_optimization

Enables additional, aggressive fixing of hold violations and maximum transition violations by the **route_opt** command.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable enables additional, aggressive fixing of hold violations and maximum transition violations by the **route_opt** command. This variable affects optimization by the **route_opt** command and accelerates design closure.

SEE ALSO

`route_opt(2)`

routeopt_enable_incremental_track_assign

Controls whether the **route_opt** command performs full or incremental track assignment when Zroute is the default router and you use the **-stage track** or **-effort high** options.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable controls whether the **route_opt** command performs full or incremental track assignment when Zroute is the default router and you use the **-stage track** or **-effort high** options.

By default (**false**), when the **route_opt** command performs track assignment, it does a full rip-up and reroute.

When you set this variable to **true**, and use the **-stage track** or **-effort high** options with the **route_opt** command, Zroute performs incremental track assignment for nets that have already had tracks assigned.

You should use this variable when you have a track-assigned database and you want to make minimal changes to the current track assignment, which could help with convergence issues for some designs.

SEE ALSO

`route_opt(2)`

routeopt_leakage_over_drc

Specifies the cost priority between the leakage cost and the maximum design rule cost during leakage recovery by the **route_opt** command.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable specifies the cost priority between the leakage cost and the maximum design rule cost during leakage recovery by the **route_opt** command.

When **false** (the default), the maximum design rule cost has a higher priority than the leakage cost.

When set to **true**, the leakage cost has a higher priority than the maximum design rule cost.

This variable works only during the leakage recovery stage; it does not have any impact in any other optimization stage.

SEE ALSO

`route_opt(2)`

routeopt_leakage_over_hold

Specifies the cost priority between the leakage cost and the minimum delay cost during leakage recovery by the **route_opt** command.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable specifies the cost priority between the leakage cost and the minimum delay cost during leakage recovery by the **route_opt** command.

When **false** (the default), the minimum delay cost has a higher priority than the leakage cost.

When set to **true**, the leakage cost has a higher priority than the minimum delay cost.

This variable works only during the leakage recovery stage; it does not have any impact in any other optimization stage.

SEE ALSO

`route_opt(2)`

routeopt_leakage_over_setup

Specifies the cost priority between the leakage cost and the maximum delay cost during leakage recovery by the **route_opt** command.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable specifies the cost priority between the leakage cost and the maximum delay cost during leakage recovery by the **route_opt** command.

When **false** (the default), the maximum delay cost has a higher priority than the leakage cost.

When set to **true**, the leakage cost has a higher priority than the maximum delay cost.

This variable works only during the leakage recovery stage; it does not have any impact in any other optimization stage.

SEE ALSO

`route_opt(2)`

routeopt_max_parallel_computations

This variable is used to override set_host_options for running routeopt in multi-thread mode.

TYPE

Integer

DEFAULT

0

GROUP

routeopt_variables

DESCRIPTION

The default value is 0, and in this mode, set_host_options -max_cores will control the multi-thread behavior of routeopt. This variable only has an effect if set_host_options -max_cores is specified. Otherwise, it is ignored. If it is set to 1, routeopt will run in a single-thread regardless of the number of cores that the set_host_options command specifies. Likewise, if it is set to a number greater than 1, routeopt will run in multi-thread mode with that number of threads, as long as it is less than or equal to the set_host_options specified value. Values specified by this variable which are greater than the latter will be ignored. This variable should not be used for normal operation and is primarily intended for diagnostic purposes. This variable replaces routeopt_mt_num (obsolete).

EXAMPLE

```
set_host_options -max_cores 4             # runs with 4 cores

set_host_options -max_cores 4             # runs with 2 cores
set routeopt_max_parallel_computations 2

set_host_options -max_cores 4             # runs in single thread
set routeopt_max_parallel_computations 1

set_host_options -max_cores 2             # runs with 2 cores
set routeopt_max_parallel_computations 4
```

SEE ALSO

`set_host_options(2)`

routeopt_only_size_weak_drive_cells

Directs the **route_opt -incremental** command to perform only sizing of weak drive cells during optimization; it will not try to fix other costs.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable directs the **route_opt -incremental** optimization to improve the drive strength of weak cells and restricts the command to size-only optimization.

When this variable is set to **true**, the **route_opt** command tries to size the design to generate a robust design with similar or better QoR that is less sensitive to variations. This feature prevents QoR degradation due to variations, such as signoff settings and parasitic changes, in the flow.

The **route_opt** command scans the design and finds sensitive cells with low drive strengths and small sizes. It sizes them to reduce setup QoR sensitivity. Maximum setup critical cells within a predefined slack range of 250ps from the worst negative slack (WNS) of its path group are selected for sizing. Sizing is restricted to footprint sizing.

This can be used as a final step before going to signoff.

SEE ALSO

`route_opt(2)`

routeopt_power_fanout_threshold

Specifies the maximum number of loads for a driver to be a candidate for leakage recovery during postroute optimization.

TYPE

integer

DEFAULT

-1

GROUP

routeopt_variables

DESCRIPTION

During leakage recovery in postroute optimization, the tool filters out some cells if they can hurt convergence and quality of results (QoR). The filtering is done based on multiple categories, one of which is the number of fanouts. If a driver has many loads, it has high chance of disturbing the routing, which results in disturbing the QoR. The tool adaptively adjusts the fanout threshold based on the drive strength and library characteristics. You can override the threshold by setting this variable.

SEE ALSO

`route_opt(2)`

routeopt_preserve_routes

Controls whether the slight routing disturbances caused by postroute optimization are reconnected in the appropriate metal layers and written to the database.

TYPE

Boolean

DEFAULT

true

GROUP

routeopt_variables

DESCRIPTION

When the **route_opt** command changes a cell during optimization, the net identities can change. For example, inserting a buffer creates a new net. Although the **route_opt** optimization seeks to place new cells along the route and maintain the route topology, it does not simultaneously reassign the net identity of the route metalization geometries associated with the new net. Route preservation performs this task in a process executed after the optimization phase finishes and before ECO routing begins. Small segments are also inserted, where necessary, to reconnect the pins displaced by legalization. The presence of the route segments bound to the nets assists the ECO router to find a similar solution to the one obtained before optimization. In this way, preserving the routes improves the correlation between the pre- and post-optimization routing and therefore acts to improve the timing convergence before and after ECO.

When this variable is set to **false**, the route shapes stay with the net that takes the original name and the routes are removed from the other nets. ECO routing performs all of the reconnections, but it can arrive at a significantly different solution, including different topology, metalization layers, and detours, than the preoptimization one.

SEE ALSO

`route_opt(2)`

routeopt_restrict_tns_to_size_only

Restricts total negative slack (TNS) optimization by the **route_opt** command to size-only optimization.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable restricts TNS optimization by the **route_opt** command to size-only optimization. This variable allows a restricted and convergent **route_opt** step. When this variable is set to **true**, worst negative slack (WNS), hold time, and design-rule fixing is done aggressively, but TNS optimization is restricted to size-only transforms.

This variable does not impact the **focal_opt** command.

SEE ALSO

`route_opt(2)`

routeopt_skip_report_qor

Skip QoR reports in the last stage during the route_opt

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

When the variable is set to true, route_opt will skip QoR reports in the last stage, and print the information "Skip Report QoR" instead.

SEE ALSO

`route_opt(2)`

routeopt_verbose

Enables verbose output for information related to route_opt operations.

TYPE

positive integer

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

When this variable is set to a positive value, the tool outputs additional information during optimization, including more information about bufferability, reasons why certain violations are not fixed, and don't touch information.

This variable applies only to hold fixing, DRC fixing, crosstalk reduction stages, and the **focal_opt** command.

The **routeopt_verbose** variable is operated by bitwise operation and you can enable verbose output per optimization engine:

Bit 0: General information

Bits 1 and 2: DRC fixing engine

- Bit 1: Terse mode (Prints a concise version of messages)
- Bit 2: Verbose mode (Enables very verbose messages)

Bits 3 and 4: Hold fixing

- Bit3: Terse mode
- Bit4: Verbose mode

Bits 5 and 6: Crosstalk reduction

- Bit3: Terse mode
- Bit4: Verbose mode

SEE ALSO

```
route_opt(2)
focal_opt(2)
```

routeopt_xtalk_reduction_cell_sizing

Turn on sizing optimization for crosstalk fixing during post route optimization.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

When this variable is set to true, the **route_opt** crosstalk reduction stage will include a sizing (footprint-preserved sizing only) optimization stage to fix crosstalk violations. This is done before and in addition to the current routing based crosstalk reduction fixing using NDR rules.

SEE ALSO

`route_opt(2)`

routeopt_xtalk_reduction_setup_threshold

The threshold of setup delta delay to perform cross talk reduction

TYPE

Float

DEFAULT

0

GROUP

routeopt_variables

DESCRIPTION

The nets with delta delay more than this threshold are selected for crosstalk reduction during route_opt. The threshold MUST be specified in "ns" irrespective of library time units.

NOTE

This variable has been deprecated since the 2010.03 release if zroute (default router) is used. However, it still applies if user has chosen to run with the classic router.

SEE ALSO

`route_opt(2)`

routing_corridor_attributes

Contains attributes related to routing corridor.

DESCRIPTION

Contains attributes related to routing corridor.

You can use **get_attribute** to determine the value of an attribute, and use **report_attribute** to get a report of all attributes on the specified object. Specified with **list_attribute -class routing_corridor -application**, the definition of attributes can be listed.

Routing Corridor Attributes

name

Specifies name of a routing corridor object.

The data type of **name** is string.

This attribute is read-only.

nets

Specifies the nets associated with the routing corridor.

The data type of **nets** is collection.

This attribute is read-only.

object_class

Specifies object class name of a routing corridor, which is **routing_corridor**.

The data type of **object_class** is string.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

routing_corridor_shape_attributes

Attributes related to routing corridor shapes.

DESCRIPTION

This man page describes the attributes related to routing corridor shapes.

To list the definitions of the routing corridor shape attributes, use the **list_attributes -application -class routing_corridor_shape** command.

To determine the value of an attribute, use the **get_attribute** command. To get a report of all attributes on a specified object, use the **report_attribute** command.

Routing Corridor Shape Attributes

bbox

Specifies the bounding-box of a routing corridor shape, which is represented by a rectangle.

The format of a rectangle specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

max_layer

Specifies layer name for the maximum layer constraint for this shape.

The data type of **max_layer** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

min_layer

Specifies layer name for the minimum layer constraint for this shape.

The data type of **min_layer** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

name

Specifies the name of a routing corridor shape.

The data type of **name** is string.

This attribute is read-only.

object_class

Specifies object class name of a routing corridor shape, which is **routing_corridor_shape**.

The data type of **object_class** is string.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

rp_group_attributes

Contains attributes that can be placed on a relative placement group.

DESCRIPTION

Contains attributes that can be placed on a relative placement group.

To set an attribute, use the command identified in the individual description of that attribute. If an attribute is read-only, you cannot set it.

To determine the value of an attribute, use the **get_attribute** command. To remove attributes, use the **remove_attribute** command.

For a more detailed explanation of an attribute, see the man page of the appropriate **set** command. For information about all attributes, see the **attributes** man page.

RP Group Attributes

alignment

Specifies the default alignment method to use when placing leaf cells and relative placement groups in the specified relative placement groups. If you do not specify this option, the tool uses bottom-left alignment.

You can set this attribute by using the **set_rp_group_options** command.

allow_non_rp_cells

Specifies that the hard keepout in the relative placement group should be overlapped with tap cells if needed. By default, the value is **false**.

You can set this attribute by using the **set_rp_group_options** command.

anchor_corner

Specifies the corner for the anchor point that is set by using the **-x_offset** and **-y_offset** options.

Valid values are **bottom-left** (the default), **bottom-right**, **top-left**, **top-right**, and **rp-location**.

If you specify **bottom-left**, the anchor point corner is the lower-left corner of the relative placement group.

If you specify **bottom-right**, the anchor point corner is the lower-right corner of the relative placement group.

If you specify **top-left**, the anchor point corner is the upper-left corner of the relative placement group.

If you specify **top-right**, the anchor point corner is the upper-right corner of the relative placement group.

If you specify **rp-location**, the anchor point corner is the starting location of the object at the row and column specified by the **-anchor_row** and **-anchor_columnoptions.**

This option applies only to top-level relative placement groups and is ignored for hierarchical relative placement groups. When you specify **-anchor_corner bottom-right**, the relative placement group is anchored at the bottom-right corner at the specified x- and y-coordinates during legalization. When many blockages are present, a slight deviation from the anchor point might occur.

You can set this attribute by using the **set_rp_group_options** command.

cell_orient_opt

A Boolean value that specifies if cell orientation optimization is done for the cells of the relative placement group for optimizing wire-length. This is a read-only attribute and cannot be modified by the user.

columns

An integer value that specifies the number of columns of the specified relative placement group.

This is a read-only attribute and cannot be modified by the user.

compress

A Boolean variable that specifies if compression is set in the horizontal direction to a relative placement group during placement. Setting this option places each row of a relative placement group without any gaps between leaf cells, lower-level hierarchical relative placement groups, or keepouts. Column alignment is not maintained when you use the **-compress** option.

You can set this attribute by using the **set_rp_group_options** command.

is_top

A Boolean value that specifies whether a relative placement group is the top.

This is a read-only attribute and cannot be modified by the user.

cts_option

Specifies how to treat the cells in the relative placement group during clock tree synthesis and clock tree optimization.

Valid values are **fixed_placement** (the default) and **size_only**.

If you specify **fixed_placement**, the cells in the relative placement group are treated as fixed during the **compile_clock_tree**, **optimize_clock_tree**, and **clock_opt** commands and cannot be sized or moved.

If you specify **size_only**, the cells in the relative placement group can only be sized. This option is applicable to the **compile_clock_tree**, **optimize_clock_tree**, and **clock_opt** commands.

You can set this attribute by using the **set_rp_group_options** command.

group_orient

A string that specifies the user-specified orientation that is set on the relative placement group.

Valid values are **default**, **N** (north), **S** (south), **FN** (flip-north), and **FS** (flip-south). The default is **N**.

You can set this attribute by using the **set_rp_group_options** command.

ignore

A Boolean value that specifies whether a relative placement group is ignored.

You can set this attribute by using the **set_rp_group_options** command.

move_effort

A string that specifies the move effort of relative placement group.

Valid values are **high**, **medium** (the default), and **low**.

You can set this attribute by using the **set_rp_group_options** command.

name

A string that specifies the full name of a relative placement group.

This is a read-only attribute and cannot be modified by the user.

pin_align_name

A string the specifies the name of pin on which the relative placement cells will be aligned.

You can set this attribute by using the **set_rp_group_options** command.

placement_type

A string that specifies the type of placement of a relative placement group.

You can set this attribute by using the **set_rp_group_options** command.

psynopt_option

Specifies the behavior of the relative placement cells of the specified relative placement group during the **psynopt** command.

You can set this attribute by using the **set_rp_group_options** command.

route_opt_option

Specifies the behavior of the relative placement cells of the specified relative placement group during the **route_opt** command.

You can set this attribute by using the **set_rp_group_options** command.

rows

An integer value that specifies the number of rows of a relative placement group.

This is a read-only attribute and cannot be modified by the user.

rp_height

A string that specifies the height of a relative placement group. It contains an estimated height before placement and the actual height if the relative placement group has gone through any kind of placement.

This is a read-only attribute and cannot be modified by the user.

rp_width

A string that specifies the width of a relative placement group. It contains an estimated width before placement and the actual width if the relative placement group has gone through any kind of placement.

This is a read-only attribute and cannot be modified by the user.

placed_orient

A string that specifies the placed orientation of a relative placement group.

Valid values are **N**, **S**, **FN**, and **FS**. If the relative placement group is not placed, it reports that the relative placement group is not placed.

This is a read-only attribute and cannot be modified by the user.

is_placed

A Boolean value that specifies whether a relative placement group is placed.

If the relative placement group is placed without any critical failures, it is set to **true**; otherwise, it is set to **false**.

This is a read-only attribute and cannot be modified by the user.

utilization

A floating point nonzero, positive value that specifies the area utilization of a relative placement group.

The default 1. The maximum value is 1.

You can set this attribute by using the **set_rp_group_options** command.

x_offset

A floating point value that specifies the x-coordinate of a relative placement group's anchor point.

You can set this attribute by using the **set_rp_group_options** command.

y_offset

A floating point value that specifies the y-coordinate of a relative placement group's anchor point.

You can set this attribute by using the **set_rp_group_options** command.

anchor_row

An integer value that specifies the row of the object for which the rp-location anchor corner is specified.

You can set this attribute by using the **set_rp_group_options** command.

anchor_column

An integer value that specifies the column of the object for which the rp-location anchor corner is specified.

You can set this attribute by using the **set_rp_group_options** command.

sdc_write_unambiguous_names

Ensures that cell, net, pin, lib_cell, and lib_pin names that are written to the SDC file are not ambiguous.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable ensures that cell, net, pin, lib_cell, and lib_pin names that are written to the SDC file are not ambiguous.

When the hierarchy has been partially flattened, embedded hierarchy separators can make names ambiguous. It is unclear which hierarchy separator characters are part of the name and which are real separators.

Beginning with SDC Version 1.2, hierarchical names can be made unambiguous using the **set_hierarchy_separator** SDC command and/or the **-hsc** option for the **get_cells**, **get_lib_cells**, **get_lib_pins**, **get_nets**, and **get_pins** SDC object access commands. By default, the tools write an SDC file, using these features to create unambiguous names.

It is wise to write SDC files that contain names that are not ambiguous. However, if you are using a third-party application that does not fully support SDC 1.2 or later versions (that is, it does not support the unambiguous hierarchical names features of SDC), you can suppress these features by setting the variable **sdc_write_unambiguous_names** to **false**. The **write_sdc** command issues a warning if you have set this variable to **false**.

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

SEE ALSO

```
printvar(2)
write_sdc(2)
```

sdfout_allow_non_positive_constraints

Writes out PATHCONSTRAINT constructs with nonpositive (<= 0) constraint values. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

Boolean

DEFAULT

false

GROUP

links_to_layout_variables

DESCRIPTION

Writes out PATHCONSTRAINT constructs with nonpositive (<= 0) constraint values. When *true*, **write_constraints -format sdf** writes out PATHCONSTRAINT constructs with nonpositive (<= 0) constraint values. When *false* (the default), paths with nonpositive constraints are written with a constraint value of 0.01.

Nonpositive constraints can occur when the arrival time at a path startpoint is larger than the required arrival time at the path endpoint. This typically indicates an error, but is sometimes valid when generating constraints for a subdesign.

To determine the current value of this variable, type **printvar sdfout_allow_non_positive_constraints**. For a list of all **links_to_layout** variables and their current values, type **print_variable_group links_to_layout**.

SEE ALSO

sdfout_min_fall_cell_delay

Specifies the minimum non-back-annotated fall cell delay that the **write_timing** command writes to a timing file in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

0

GROUP

io_variables

DESCRIPTION

Specifies the minimum non-back-annotated fall cell delay that the **write_timing** command writes to a timing file in SDF format. The value of this variable can be *positive*, *negative*, or *zero* (the default); the unit must be the same as the timing unit in the technology library.

By default, if this variable is not set, **write_timing** writes to the SDF file all fall cell delay values greater than or equal to zero. You can override this default behavior for non-back-annotated delays by setting **sdfout_min_fall_cell_delay** to a minimum value; **write_timing** does not write values that are less than this minimum. However, you cannot override the default behavior for back-annotated delays; **write_timing** always writes values of delays that have been back-annotated, regardless of the value of this variable.

Use this variable to filter non-back-annotated delays so that **write_timing** writes only values that are significant (greater than the specified minimum value). Also, if you do not want non-annotated fall cell delays to be written to the SDF file, set this variable to a value higher than any of the non-annotated cell delays in the design.

To determine the current value of this variable, use **printvar sdfout_min_fall_cell_delay**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

`sdfout_min_rise_cell_delay(3)`

sdfout_min_fall_net_delay

Specifies the minimum non-back-annotated fall net delay that **write_timing** can write to a timing file in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

0

GROUP

io_variables

DESCRIPTION

Specifies the minimum non-back-annotated fall net delay that **write_timing** can write to a timing file in SDF format. The value of this variable can be *positive*, *negative*, or *zero* (the default); the unit must be the same as the timing unit in the technology library.

By default, if this variable is not set, **write_timing** writes to the SDF file all fall net delay values greater than or equal to zero. You can override this default behavior for non-back-annotated delays by setting **sdfout_min_fall_net_delay** to a minimum value; **write_timing** does not write values that are less than this minimum. However, you cannot override the default behavior for back-annotated delays; **write_timing** always writes values of delays that have been back-annotated, regardless of the value of this variable.

Use this variable to filter non-back-annotated delays so that **write_timing** writes only values that are significant (greater than the specified minimum value). Also, if you do not want any non-annotated fall net delays to be written to the SDF file, set this variable to a value higher than any of the non-annotated net delays in the design.

To determine the current value of this variable, use **printvar sdfout_min_fall_net_delay**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

`sdfout_min_rise_net_delay(3)`

sdfout_min_rise_cell_delay

Specifies the minimum non-back-annotated rise cell delay that **write_timing** can write to a timing file in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

0

GROUP

io_variables

DESCRIPTION

Specifies the minimum non-back-annotated rise cell delay that **write_timing** can write to a timing file in SDF format. The value of this variable can be *positive*, *negative*, or *zero* (the default); the unit must be the same as the timing unit in the technology library.

By default, if this variable is not set, **write_timing** writes to the SDF file all rise cell delay values greater than or equal to zero. You can override this default behavior for non-back-annotated delays by setting **sdfout_min_rise_cell_delay** to a minimum value; **write_timing** will not write values less than this minimum. However, you cannot override the default behavior for back-annotated delays; **write_timing** always writes values of delays that have been back-annotated, regardless of the value of this variable.

Use this variable to filter non-back-annotated delays so that **write_timing** writes only values that are significant (greater than the specified minimum value). Also, if you do not want any non-annotated rise cell delays to be written to the SDF file, set this variable to a value higher than any of the non-annotated cell delays in the design.

To determine the current value of this variable, use **printvar sdfout_min_rise_cell_delay**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

`sdfout_min_fall_cell_delay(3)`

sdfout_min_rise_net_delay

Specifies the minimum non-back-annotated rise net delay that the **write_timing** command can write to a timing file in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

0

GROUP

io_variables

DESCRIPTION

Specifies the minimum non-back-annotated rise net delay that the **write_timing** command can write to a timing file in SDF format. The value of this variable can be *positive*, *negative*, or *zero* (the default); the unit must be the same as the timing unit in the technology library.

By default, if this variable is not set, **write_timing** writes to the SDF file all rise net delay values greater than or equal to zero. You can override this default behavior for non-back-annotated delays by setting **sdfout_min_rise_net_delay** to a minimum value; **write_timing** does not write values less than this minimum. However, you cannot override the default behavior for back-annotated delays; **write_timing** always writes values of delays that have been back-annotated, regardless of the value of this variable.

Use this variable to filter non-back-annotated delays so that **write_timing** writes only values that are significant (greater than the specified minimum value). Also, if you do not want any non-annotated rise net delays to be written to the SDF file, set this variable to a value higher than any of the non-annotated net delays in the design.

To determine the current value of this variable, use **printvar sdfout_min_rise_net_delay**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

`sdfout_min_fall_net_delay(3)`

sdfout_time_scale

Specifies the time scale of the delays written to timing files in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

float

DEFAULT

1

GROUP

io_variables

DESCRIPTION

Specifies the time scale of the delays written to timing files in SDF format. Delays from Design Compiler are written to timing files with **write_timing**. The **sdfout_time_scale** variable must be set if the library has no time unit specified and if the time unit of the delays in the library is different than 1 nanosecond. By default, the time unit is nanosecond and the time scale is 1. The only valid values for the SDF format are 0.001, 0.01, 0.1, 1, 10 and 100. The time unit is specified in the library with the attributes **time_scale** and **time_unit_name**.

For example, a library with timing values in 10 picoseconds is specified with the attributes:

time_scale = 10 and time_unit_name = ps

When the attribute **time_scale** is missing in the library, use the **sdfout_time_scale** variable to specify the scale of the timing unit. For example, if the library has no time unit specified but is in 10 ns, set **sdfout_time_scale** to 10 in order to specify the time unit as 10 ns.

To determine the current value of this variable use **printvar sdfout_time_scale**. For a list of all **io_variables** and their current values, use the **print_variable_group io** command.

SEE ALSO

sdfout_top_instance_name

Specifies the name prepended to all instance names when writing timing files in SDF format. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

string

DEFAULT

""

GROUP

io_variables

DESCRIPTION

Specifies the name prepended to all instance names when writing timing files in SDF format. Timing files are written with the **write_timing** command. By default **write_timing** prepends no name to all cell instance names. Set this variable when you want the cell instance names to contain a prepended name.

For example set **sdfout_top_instance_name = "stim.cell1"** if the timing file should contain:

```
(DESIGN "fifo")
(DIVIDER .)
(CELL
   (CELLTYPE "fifo")
   (INSTANCE stim\.cell1)
)
(CELL
   (CELLTYPE "AND2")
   (INSTANCE stim\.cell1.U1)
```

With the previous example, if **sdfout_top_instance_name = ""** timing file will contain:

```
(DESIGN "fifo")
(DIVIDER .)
(CELL
```

```
   (CELLTYPE "fifo")
   (INSTANCE )
)
(CELL
   (CELLTYPE "AND2")
   (INSTANCE U1)
```

To determine the current value of this variable, use **printvar sdfout_top_instance_name**. For a list of all **io_variables** variables and their current values, use the **print_variable_group io** command.

SEE ALSO

sdfout_write_to_output

Specifies whether the **write_timing -f sdf** command writes interconnect delays between cells and top-level output ports. This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

Boolean

DEFAULT

false

GROUP

io_variables

DESCRIPTION

Specifies whether the **write_timing -f sdf** command writes interconnect delays between cells and top-level output ports. The **sdfout_write_to_output** variable also determines whether output to output pin IOPATH statements are written for cells that contain output-to-output timing arcs. v1.0 SDF does not support output-to-output timing for either IOPATH or INTERCONNECT statements. However, the Synopsys Simulator does support output-to-output timing for these statements.

Set this variable to *true*, if the targeted SDF reader is the Synopsys Simulator.

Leave this variable set to *false*, the default, to ensure that generated SDF files comply with the v1.0 specification.

Set this variable before using **write_timing**. To check the value of this variable, use the command **printvar sdfout_write_to_output**.

SEE ALSO

search_path

Specifies directories that the tool searches for files specified without directory names.

TYPE

list

DEFAULT

{*search_path* + .}

GROUP

system_variables

DESCRIPTION

This variable specifies directories that the tool searches for files specified without directory names. The search includes looking for technology and symbol libraries, design files, and so on. The value of this variable is a list of directory names and is usually set to a central library directory.

To determine the current value of this variable, use the **printvar search_path** command. For a list of all system variables and their current values, use the **print_variable_group system** command.

SEE ALSO

system_variables(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

true

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

Suppresses CMD-104 messages for variables in this list.

TYPE

string

DEFAULT

""

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_arch

Indicates the system architecture of your machine.

TYPE

string

DEFAULT

linux64

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

Anywhere

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

true

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 is written a log of the initial values of variables and executed commands.

TYPE

string

DEFAULT

command.log

DESCRIPTION

Specifies the name of the file to which is written a log of the initial values of variables and executed commands.

By default, the tool writes the log to a file named command.log. If the value is an empty string, a command log file is not created.

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

SEE ALSO

```
printvar(2)
view_command_log_file(3)
view_log_file(3)
```

sh_continue_on_error

Allows processing to continue when errors occur during script execution with the **source** command.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable is deprecated. It is recommended to use the **-continue_on_error** option to the **source** command instead of this variable because that option only applies to a single script, and not the entire application session.

When set to **true**, the **sh_continue_on_error** variable allows processing to continue when errors occur. Under normal circumstances, when executing a script with the **source** command, Tcl errors (syntax and semantic) cause the execution of the script to terminate.

When **sh_continue_on_error** is set to **false**, script execution can also terminate due to new error and warning messages based on the value of the **sh_script_stop_severity** variable.

To determine the current value of the **sh_continue_on_error** variable, use the **get_app_var sh_continue_on_error** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_script_stop_severity(3)
```

sh_deprecated_is_error

Raise a Tcl error when a deprecated command is executed.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set this variable causes a Tcl error to be raised when an deprecated command is executed. Normally only a warning message is issued.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
```

sh_dev_null

Indicates the current null device.

TYPE

string

DEFAULT

/dev/null

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_stdout_redirect

Allows the redirect command to capture output to the Tcl stdout channel.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When set to **true**, this variable allows the redirect command to capture output sent to the Tcl stdout channel. By default, the Tcl **puts** command sends its output to the stdout channel.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
```

sh_help_shows_group_overview

Changes the behavior of the "help" command.

TYPE

string

DEFAULT

true

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_new_variable_message

Controls a debugging feature for tracing the creation of new variables.

TYPE

Boolean

DEFAULT

true

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_varcommand 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)
set_app_var(2)
```

```
sh_new_variable_message(3)
sh_new_variable_message_in_script(3)
```

sh_new_variable_message_in_script

Controls a debugging feature for tracing the creation of new variables within a sourced script.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **sh_new_variable_message_in_script** variable controls a debugging feature for tracing the creation of new variables within a sourced script. Its primary debugging purpose is to catch the misspelling of an application-owned global variable.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. See the **set_app_var** command man page for details.

Note that the **sh_new_variable_message** variable must be set to **true** for this variable to have any effect. Both variables must be set to **true** for the feature to be enabled. In that case, an informational message (CMD-041) is displayed when a variable is defined for the first time. When **sh_new_variable_message_in_script** is set to **false** (the default), no message is displayed at the time that the variable is created. When the **source** command completes, however, you see messages for any new variables that were created in the script. This is because the state of the variables is sampled before and after the **source** command. It is not because of inter-command sampling within the script. So, this is actually a more efficient method to see if new variables were created in the script.

For example, given the following script a.tcl:

```
echo "Entering script"
set a 23
echo a = $a
set b 24
echo b = $b
echo "Exiting script"
```

When **sh_new_variable_message_in_script** is **false** (the default), you see the following when you source the script:

```
prompt> source a.tcl
Entering script
a = 23
b = 24
Exiting script
Information: Defining new variable 'a'. (CMD-041)
Information: Defining new variable 'b'. (CMD-041)
prompt>
```

Alternatively, when **sh_new_variable_message_in_script** is **true**, at much greater cost, you see the following when you source the script:

```
prompt> set sh_new_variable_message_in_script true
Warning: Enabled new variable message tracing -
         Tcl scripting optimization disabled. (CMD-042)
true
prompt> source a.tcl
Entering script
Information: Defining new variable 'a'. (CMD-041)
a = 23
Information: Defining new variable 'b'. (CMD-041)
b = 24
Exiting script
prompt>
```

Warning: This feature has a significant negative impact on CPU performance. This should be used only when developing scripts or in interactive use. When you turn on the feature, the application issues a message (CMD-042) to warn you about the use of this feature.

To determine the current value of this variable, use the **get_app_var sh_new_variable_message_in_script** command.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
sh_new_variable_message(3)
sh_new_variable_message_in_proc(3)
```

sh_obsolete_is_error

Raise a Tcl error when an obsolete command is executed.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set this variable causes a Tcl error to be raised when an obsolete command is executed. Normally only a warning message is issued.

Obsolete commands have no effect.

SEE ALSO

```
get_app_var(2)
set_app_var(2)
```

sh_output_log_file

This variable is used to name the file to record console output. Set the variable to the empty string to disable output capture.

TYPE

String

DEFAULT

icc_output.txt

DESCRIPTION

This variable can be used to save almost all console output to a log file. The log file is useful for bug reproduction and reporting.

The first time the variable is set to a valid filename, all previously logged output is written to the specified file, overwriting any previous contents of the file. All subsequent logged output is appended to the file unless the variable is later reset.

If the variable is later changed to an empty string, all logged output is disabled.

If the variable is later set to an non-empty string that is an invalid filename, the new value is ignored and the old value is restored.

If the variable is later set to an non-empty string that is an valid filename, all subsequent logged output is written to this file. Any previous contents of the file are overwritten.

The log file may not capture the banner text from the beginning of the session, and does not capture the stack trace that is printed following a fatal error. You can use the UNIX tee command or redirect the output to capture this information.

This variable is not available in de_shell mode.

SEE ALSO

`printvar(2)`

sh_product_version

Indicates the version of the application currently running.

TYPE

string

DESCRIPTION

This variable is set to the version of the application currently running. The variable is read only.

To determine the current value of this variable, use the **get_app_var sh_product_version** command.

SEE ALSO

`get_app_var(2)`

sh_script_stop_severity

Indicates the error message severity level that would cause a script to stop running before it completes.

TYPE

string

DEFAULT

None

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

None

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

true

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

Causes the **search** command to use the **search_path** variable to search for files. This variable is for use in **dc_shell-t** (Tcl mode of dc_shell) only.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

Causes the **search** command to use the **search_path** variable to search for files, when **sh_source_uses_search_path** is *true* (the default). When *false*, the **source** command considers this variable's file argument literally. This variable is for use in **dc_shell-t** (Tcl mode of dc_shell) only.

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

SEE ALSO

```
printvar(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

""

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)
```

shape_attributes

Attributes related to shapes.

DESCRIPTION

This man page describes attributes related to shapes.

To list the definitions of the shape attributes, use the **list_attribute -application -class shape** command.

To determine the value of an attribute, use the **get_attribute** command.

To report all attributes on a specified object, use the **report_attribute** command.

Shape Attributes

bbox

Specifies the bounding-box of a shape.

The **bbox** attribute is represented by a rectangle. The format of a rectangle specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

bbox_ll

Specifies the lower-left corner of the bounding-box of a shape.

The **bbox_ll** attribute is represented by a point. The format of a point specification is {x y}.

You can get the **bbox_ll** attribute of a shape by accessing the first element of its **bbox** attribute.

The data type of **bbox_ll** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

bbox_llx

Specifies the x-coordinate of the lower-left corner of the bounding-box of a shape.

The data type of **bbox_llx** is integer.

This attribute is read-only.

bbox_lly

Specifies the y-coordinate of the lower-left corner of the bounding-box of a shape.

The data type of **bbox_lly** is integer.

This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a shape.

The \fbbox_ur attribute is represented by a point. The format of a point specification is {x y}.

You can get the **bbox_ur** attribute of a shape by accessing the second element of its **bbox** attribute.

The data type of **bbox_ur** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

bbox_urx

Specifies the x-coordinate of the upper-right corner of the bounding-box of a shape.

The data type of **bbox_urx** is integer.

This attribute is read-only.

bbox_ury

Specifies the y-coordinate of the upper-right corner of the bounding-box of a shape.

The data type of **bbox_ury** is integer.

This attribute is read-only.

cell_id

Specifies the Milkyway design ID in which a shape object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

datatype_number

Specifies the GDSII datatype number of a shape object.

The data type of **datatype_number** is integer.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

double_pattern_mask_constraint

Specifies the coloring information for a shape. This information is needed in the double-patterning flow.

The data type of **double_pattern_mask_constraint** is string.

Its valid values are

- **any_mask**
- **mask1_soft**
- **mask1_hard**
- **mask2_soft**
- **mask2_hard**
- **same_mask**

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

endcap

Specifies the alignment type for the end of a wire or path shape.

The data type of **endcap** is string.

Its valid values are

- **square_ends**
- **round_ends**
- **square_ends_by_half_width**
- **octagon_ends_by_half_width**

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

layer

Specifies the layer name of a shape.

The data type of **layer** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

layer_name

Specifies the layer name of a shape.

The data type of **layer_name** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

layer_number

Specifies the layer number of a shape.

The data type of **layer_number** is integer.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

length

Specifies the length of a wire or path shape in user units.

The data type of **length** is float.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

multiple_pattern_mask_constraint

Specifies the coloring information for a shape. This information is needed in the multiple-patterning flow.

The data type of **multiple_pattern_mask_constraint** is string.

Its valid values are

- **any_mask**
- **mask1_soft**
- **mask1_hard**
- **mask2_soft**
- **mask2_hard**
- **mask3_soft**
- **mask3_hard**
- **same_mask**

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

name

Specifies the name of a shape.

The data type of **name** is string.

This attribute is read-only.

net_id

Specifies the object ID of the net associated with a shape.

The data type of **net_id** is integer.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

net_type

Specifies the type of net associated with a shape.

The data type of **net_type** is string.

This attribute is read-only.

object_class

Specifies the object class name of a shape, which is **shape**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies the object ID in the Milkyway design file.

The data type of **object_id** is integer.

This attribute is read-only.

object_type

Specifies the object type, which can be **RECTANGLE**, **POLYGON**, **TRAPEZOID**, **PATH**, **HWIRE**, or **VWIRE**.

The data type of **object_type** is string.

The attribute is read-only.

owner

Specifies the Milkyway design file name in which a shape is located.

The data type of **owner** is string.

This attribute is read-only.

owner_net

Specifies the name of the net to which a shape is connected.

The data type of **owner_net** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

points

Specifies the point list of a shape's boundary.

The data type of **points** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

route_type

Specifies the route type of a shape.

The data type of **route_type** is string.

Its valid values are

- **User Enter** or **user_enter**
- **Signal Route** or **signal_route**
- **Signal Route (Global)** or **signal_route_global**
- **P/G Ring** or **pg_ring**
- **Clk Ring** or **clk_ring**
- **P/G Strap** or **pg_strap**
- **Clk Strap** or **clk_strap**
- **P/G Macro/IO Pin Conn** or **pg_macro_io_pin_conn**
- **P/G Std. Cell Pin Conn** or **pg_std_cell_pin_conn**
- **Zero-Skew Route** or **clk_zero_skew_route**
- **Bus** or **bus**
- **Shield (fix)** or **shield**
- **Shield (dynamic)** or **shield_dynamic**
- **Fill Track** or **clk_fill_track**

- **Unknown** or **unknown**

The **mw_attr_value_no_space** variable determines whether the **get_attribute** or **report_attribute** command returns the value containing spaces or underscores.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

width

Specifies the width of a wire or path shape in user units.

The data type of **width** is float.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

si_ccs_use_gate_level_simulation

Affects the behavior of timing analysis and postroute optimization when crosstalk effects are enabled.

TYPE

Boolean

DEFAULT

false

GROUP

si_variables

DESCRIPTION

This variable affects the behavior of timing analysis and postroute optimization when crosstalk effects are enabled.

When set to **true**, it enables the use of CCS noise engine for delta delay computation. To use this feature, make sure that your library contains proper CCS noise data.

Enabling this feature improves the IC Compiler and PrimeTime SI correlation, but it can lead to runtime increase.

To determine the current value of this variable, use the **get_app_var si_ccs_use_gate_level_simulation** command.

SEE ALSO

```
report_timing(2)
si_variables(3)
```

si_max_parallel_computations

Sets the degree of parallelism used for parallel signal integrity computations.

TYPE

integer

DEFAULT

0

GROUP

si_variables

DESCRIPTION

This application variable specifies the degree of parallelism used for parallel signal integrity computations.

When the variable has a value of **1**, parallel signal integrity computation is disabled.

When the variable has a value of **0** (the default), or a value that is larger than **1**, parallel signal integrity computation is enabled.

SEE ALSO

`set_host_options(2)`

si_variables

Variables that affect signal integrity analysis and related optimization, including static noise, crosstalk delta delay and delta transition, and signal electromigration.

SYNTAX

```
Boolean si_xtalk_reselect_delta_and_slack = false
float si_xtalk_reselect_delta_delay = 5
float si_xtalk_reselect_delta_delay_ratio = 0.95
float si_xtalk_reselect_min_mode_slack = 0
float si_xtalk_reselect_max_mode_slack = 0
float si_filter_per_aggr_noise_peak_ratio = 0.01
float si_filter_accum_aggr_noise_peak_ratio = 0.03
Boolean xt_filter_logic_constant_aggressors = true
```

DESCRIPTION

These variables directly affect signal integrity analysis. Defaults are shown in the SYNTAX section.

To view an individual variable description, type **man var**, where *var* is the name of the variable.

si_xtalk_reselect_delta_and_slack

Reselect nets that satisfy both delta delay and slack reselection criteria.

si_xtalk_reselect_delta_delay

Specifies the threshold of net delay change caused by crosstalk analysis, above which the tool reselects the net for subsequent crosstalk delay calculations.

si_xtalk_reselect_delta_delay_ratio

Specifies the threshold of the ratio of net delay change caused by crosstalk analysis to the total stage delay, above which the tool reselects a net for subsequent delay calculations.

si_xtalk_reselect_min_mode_slack

Specifies the min mode pin slack threshold, below which the tool reselects a net for subsequent delay calculations.

si_xtalk_reselect_max_mode_slack

Specifies the maximum mode pin slack threshold, below which the tool reselects a net for subsequent delay calculations.

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.

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.

xt_filter_logic_constant_aggressors

Specifies if logic constant nets should be considered as aggressors in crosstalk and static noise analysis and optimization.

SEE ALSO

`set_si_options(2)`

si_xtalk_composite_aggr_noise_peak_ratio

Controls the composite aggressor selection for crosstalk analysis.

TYPE

float

DEFAULT

0.01

GROUP

si_variables

DESCRIPTION

Specifies the threshold value in crosstalk bump to VDD ratio, below which aggressors are selected into the composite aggressor group. The default is *0.01*, which means all the 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 threshold variables, **si_filter_per_aggr_noise_peak_ratio** and **si_filter_accum_aggr_noise_peak_ratio**, to determine which aggressors can be selected into the composite aggressor group.

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

```
printvar si_xtalk_composite_aggr_noise_peak_ratio
```

SEE ALSO

```
si_filter_per_aggr_noise_peak_ratio(3)
si_filter_accum_aggr_noise_peak_ratio(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

Sets the desired 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 desired 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.

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

```
prompt> printvar si_xtalk_composite_aggr_quantile_high_pct
```

SEE ALSO

`si_xtalk_composite_aggr_noise_peak_ratio(3)`

signoff_enable_dmsa_fix_hold

Enables **dmsa_fix_hold** during **signoff_opt**.

TYPE

Integer

DEFAULT

0

DESCRIPTION

The **signoff_enable_dmsa_fix_hold** variable enables **dmsa_fix_hold** during **signoff_opt**. The default value is **0**.

dmsa_fix_hold is a tcl procedure provided and maintained by **PrimeTime**, which performs hold fixing on multi-mode multi-corner designs for **PrimeTime** with **Distributed Multi-Scenario Analysis (DMSA)**. By changing the value to a positive number, you enable the feature to invoke **dmsa_fix_hold** during **signoff_opt** as a preface optimization step to the normal signoff driven optimization flow. The actual value of **signoff_enable_dmsa_fix_hold** denotes how many iterations **dmsa_fix_hold** should be invoked by **signoff_opt**. You can combine this with **-num_iteration** to control the total number of iterations performed by **signoff_opt**.

Since **IC Compiler** does not distribute the tcl procedure **dmsa_fix_hold**, you need to get access of this tcl procedure by contacting **PrimeTime**. In addition, you need to ensure:

- Enable this tcl variable
- **set signoff_enable_dmsa_fix_hold 1**
- Provide the tcl file containing the tcl procedure **dmsa_fix_hold** to **IC Compiler**
- **set signoff_pt_dmsa_script_file /mypath/dmsa_fix_hold.tcl**
- Your design must use multi-mode multi-corner setup
- Use **PrimeTime** version 2008.12 or later

During **signoff_opt**, you should then see the progress from **dmsa_fix_hold** (actual messages are subject to change by implementation of **dmsa_fix_hold**):

```
prompt> create_scenarion Func
prompt> set signoff_enable_dmsa_fix_hold 1
prompt> set signoff_pt_dmsa_script_file /mypath/dmsa_fix_hold.tcl
prompt> set_primetime_options -exec_dir /ptpath/2008.12/linux/syn/bin
prompt> signoff_opt

  ...

Beginning iteration 1...

Obtaining hold-critical timing paths...
  1000 hold-critical endpoints found within 1000 hold-critical paths.

Processing paths... Obtaining pin slacks from scenarios...

  ...
```

EXAMPLE

The following example specifies there should be exactly 4 iterations of optimization in **signoff_opt**, among which, the first 3 iterations are calls to **dmsa_fix_hold**:

```
prompt> set signoff_enable_dmsa_fix_hold 3
prompt> set signoff_pt_dmsa_script_file /mypath/dmsa_fix_hold.tcl
prompt> set_primetime_options -exec_dir /ptpath/2008.12/linux/syn/bin
prompt> signoff_opt -num_iteration 4
```

SEE ALSO

signoff_enable_dmsa_fix_signoff(3)
signoff_opt(2)

signoff_enable_dmsa_fix_signoff

Enables **dmsa_fix_signoff** during **signoff_opt**.

TYPE

Integer

DEFAULT

0

DESCRIPTION

The **signoff_enable_dmsa_fix_signoff** variable enables **dmsa_fix_signoff** during **signoff_opt**. The default value is **0**.

By changing the value to a positive number, you enable the feature to invoke a user-defined tcl procedure **dmsa_fix_signoff** during **signoff_opt** as a preface optimization step to the normal signoff driven optimization flow. The actual value of **signoff_enable_dmsa_fix_signoff** denotes how many iterations **dmsa_fix_signoff** should be invoked by **signoff_opt**. You can combine this with **-num_iteration** to control the total number of iterations performed by **signoff_opt**.

The user defined tcl procedure **dmsa_fix_signoff** is a tcl procedure that is can be run on **PrimeTime Distributed Multi-Scenario Analysis (DMSA)** master process, similar to **dmsa_fix_hold** (which is provided and maintained by **PrimeTime**), except **dmsa_fix_signoff** takes no argument.

Once enabled, the tcl procedure **dmsa_fix_signoff** will be called from **signoff_opt** as a preface optimization step, and is executed on the **PrimeTime DMSA** master process. You can make your desired netlist changes inside this tcl procedure to optimize the design. However, these changes must be able to be written out by **write_changes -format icctcl**.

You can also use **dmsa_fix_signoff** as a wrapper to **dmsa_fix_hold** which allows you to customize various settings used by **dmsa_fix_hold** such as the buffer list.

For example:

```
proc dmsa_fix_signoff {args} {

  eval "dmsa_fix_hold -prefix U -effort low -verbose {DLYX1 DLYX2 DLYX4
DLYX8 BUFX1 BUFX2 BUFX4 BUFX8}"
```

```
}
```

To use this feature, you need to ensure:

- Enable this tcl variable
- **set signoff_enable_dmsa_fix_signoff 1**
- Provide the tcl file containing the tcl procedure **dmsa_fix_signoff** to **IC Compiler**
- **set signoff_pt_dmsa_script_file /mypath/dmsa_fix_signoff.tcl**
- Netlist changes made by **dmsa_fix_signoff** must be able to be written out with **write_changes -format icctcl**
- Your design must use multi-mode multi-corner setup
- Use **PrimeTime** version 2008.12 or later

During **signoff_opt**, you should then see the progress from **dmsa_fix_signoff**.

EXAMPLE

The following example specifies there should be exactly 4 iterations of optimization in **signoff_opt**, among which, the first 3 iterations are calls to **dmsa_fix_signoff**:

```
prompt> set signoff_enable_dmsa_fix_signoff 3
prompt> set signoff_pt_dmsa_script_file /mypath/dmsa_fix_hold.tcl
prompt> set_primetime_options -exec_dir /ptpath/2008.12/linux/syn/bin
prompt> signoff_opt -num_iteration 4
```

SEE ALSO

```
signoff_enable_dmsa_fix_hold(3)
signoff_opt(2)
```

signoff_enable_primetime_reselection

Controls whether additional net reselection is enabled in PrimeTime-SI during leakage optimization by the **signoff_opt** command.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

You can use the **signoff_enable_primetime_reselection** variable to enable or disable additional net reselection in PrimeTime-SI during leakage optimization by the **signoff_opt** command.

The default setting is **true**, which enables net reselection during leakage optimization by the **signoff_opt** command. Note that this variable takes effect only when the **-power** or **-only_power_recovery** options are used with the **signoff_opt** command.

When enabled, during leakage optimization by the **signoff_opt** command, an additional set of nets is forced to be reselected in PrimeTime-SI by invoking the **set_si_delay_analysis -reselect netname** PrimeTime command.

With an additional set of nets being reselected by PrimeTime-SI, the final timing results from PrimeTime-SI might be different from runs without those reselected nets.

The set of reselected nets is written out to a unique file in the run directory. The PSYN-705 message displays the name of the file.

For multicorner-multimode designs, the set of reselected nets might be different in each scenario. In this case, a different file could be written out for each scenario.

Check the PSYN-705 messages to find out which nets are forced to be reselected in PrimeTime-SI.

For subsequent runs of the **run_signoff** or **signoff_opt** command, use the **set_primetime_options -specific_file** command to utilize the files with reselected nets.

For subsequent standalone **PrimeTime-SI** runs, source these files after reading the netlist.

Setting this variable to **false** disables net reselection during leakage optimization by the **signoff_opt** command.

EXAMPLE

The following example changes the variable to **false**, and thus disables net reselection during leakage optimization in the **signoff_opt** command.

```
prompt> set signoff_enable_primetime_reselection false
prompt> signoff_opt -power
```

SEE ALSO

signoff_opt(2)
set_primetime_options(2)
PSYN-702(n)
PSYN-704(n)
PSYN-705(n)

signoff_pt_dmsa_script_file

Provides tcl script to enable **dmsa_fix_hold** or **dmsa_fix_signoff** during **signoff_opt**.

TYPE

string

DEFAULT

""

DESCRIPTION

You must use this variable together with either **signoff_enable_dmsa_fix_hold** or **signoff_enable_dmsa_fix_signoff**, to provide the tcl script file which contains the definition of tcl procedure **dmsa_fix_hold** or **dmsa_fix_signoff**.

SEE ALSO

```
signoff_opt(2)
signoff_enable_dmsa_fix_hold(3)
signoff_enable_dmsa_fix_signoff(3)
```

simplified_verification_mode

Performs optimization so that formal verification is prioritized over quality of results (QoR).

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

Setting this variable to **true** adjusts optimization inside the tool to prioritize formal verification compatibility over QoR. This enables single-pass verification for formal verification to pass with as little effort as possible. The variable is set to **false** by default.

This variable affects the optimization done during the reading and elaboration of RTL files and when the **insert_clock_gating** and **compile_ultra** commands are run.

When you set this variable to **true**, the **compile_ultra** command runs with the **-no_autoungroup** option enabled. However, the **-retime** option of the **compile_ultra** command is ignored, and the **-global** option of the **insert_clock_gating** and **replace_clock_gates** commands is ignored.

The tool sets the value for the following environment variables when the **simplified_verification_mode** variable is set to **true** regardless of the value you specify:

```
compile_ultra_ungroup_dw = false,
compile_clock_gating_through_hierarchy = false
hdlin_verification_priority = true
```

The value for these variables is restored to the user-specified value when you set the **simplified_verification_mode** variable to **false**. If you do not specify the value for these variables, the tool uses the default value.

When you run the **compile_ultra** command, the **simplified_verification_mode** variable adjusts the optimizations on datapath logic. It also identifies CRC logic in the design and isolates it by creating a new hierarchy.

When you set the **simplified_verification_mode** variable to **true**, designs with the **optimize_registers** attribute and DW_div_pipe components are not retimed because they can adversely affect verification success. However, optimizing these designs without retiming them can have a large impact on QoR and runtime. To enable the retiming of these designs, set the **simplified_verification_mode_allow_retiming** variable to **true** when you enable the **simplified_verification_mode** variable.

SEE ALSO

`simplified_verification_mode_allow_retiming(3)`

simplified_verification_mode_allow_retiming

Controls whether **DW_div_pipe** components and designs with the **optimize_registers** attribute are retimed when the **simplified_verification_mode** variable is set to **true**.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

When you set the **simplified_verification_mode** variable to **true**, designs with the **optimize_registers** attribute and DW_div_pipe components are not retimed because they can adversely affect verification success. However, optimizing these designs without retiming them can have a large impact on QoR and runtime. To enable the retiming of these designs, set the **simplified_verification_mode_allow_retiming** variable to **true** when you enable the **simplified_verification_mode** variable.

SEE ALSO

`simplified_verification_mode(3)`

single_group_per_sheet

Specifies to the tool to put only 1 logic group on a sheet.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

This variable specifies to the tool to put only 1 logic group on a sheet. Using this partitioning option set to **true** eliminates the possibility of more than 1 off-sheet connector with the same name being on a single sheet. The default value is **false**.

SEE ALSO

site_info_file

Contains the path to the site information file for licensing.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable contains the path to the site information file for licensing. The default value is the empty string.

site_row_attributes

Contains attributes related to site row.

DESCRIPTION

Contains attributes related to site row.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified site row. Specified with **list_attribute -class site_row -application**, the definition of attributes can be listed.

Site Row Attributes

cell_id

Specifies Milkyway design ID in which a site row object is located.

This attribute is read-only.

name

Specifies name of a site row object.

This attribute is read-only.

object_class

Specifies object class name of a site row object, which is **site_row**.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

This attribute is read-only.

bbox

Specifies the bbox of a site row object.

This attribute is read-only.

bbox_ll

Specifies the lower-left of a site row object.

This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower_left of the bbox a site row object.

This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower_left of the bbox a site row object.

This attribute is read-only.

bbox_ur

Specifies the upper-right of the bbox of a site row object.

This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right of the bbox a site row object.

This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right of the bbox a site row object.

This attribute is read-only.

site_type

Specifies the type of site being defined.

This attribute is read-only.

orientation

Specifies the orientation of the sites. The value can be N, W, S, E, FN, FW, FS, FE.

This attribute is read-only.

direction

Specifies the direction of the row. The value can be v, h, vertical and horizontal.

This attribute is read-only.

site_count

Specifies the number of the sites in the row.

This attribute is read-only.

site_space

Specifies the space for each site, from the lower left corner of the site to the lower left corner of the next site. The value is specified in microns.

This attribute is read-only.

origin

Specifies the lower left corner of the row, regardless of orientation.

This attribute is read-only.

allowable_pattern

Specifies the restricted orientations of placed cells on a specified row, which are based on what direction of row is, whether row is flipped and what direction of unit tile is.

This attribute is read-only.

row_type

It is just a positive integer associated with site rows. Later on set_cell_row_type can be used to associate one particular cell with some particular rows of corresponding row_type.

This attribute can be read, set and removed.

is_ignored

TRUE: ignore specified site row FALSE: The specified site row is not ignored

The attribute can be read and set.

is_reserved_placement_area

The attribute is true if the site row has the property "is_reserved_placement_area".

The attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
```

skew_opt_optimize_buffer_count

Enables or disables buffer count reduction during the skew_opt command.

TYPE

Boolean

DEFAULT

true

GROUP

skew_opt_variables

DESCRIPTION

This variable has an impact on the behavior of the skew_opt command. The skew_opt command calculates the useful skew to be applied at clock pins to improve the timing QoR (slack) of the design. The calculated useful skew is expressed as float pin exceptions that are implemented by the compile_clock_tree and optimize_clock_tree commands. When the skew_opt_optimize_buffer_count variable is set to true (the default), a trade-off is made between the estimated buffer count and timing QoR. This means that the tool allows a minor timing degradation to reduce the number of estimated clock tree buffers to implement useful skew.

With the variable set to false, the skew_opt command calculates the unconstrained useful skew for the flops in the design. In that case, the only objective of the skew_opt command is to optimize worst negative slack and total negative slack in the design. However, in some designs with many leaf-level clock gates, this might lead to many clock tree buffers in the synthesize gated clock trees.

Use the following command to determine the current value of the variable:

```
prompt> printvar skew_opt_optimize_buffer_count
```

SEE ALSO

skew_opt(2)
clock_opt(2)
compile_clock_tree(2)
optimize_clock_tree(2)

skew_opt_optimize_clock_gates

Enables clock gate (ICG) optimization during the skew_opt command.

TYPE

Boolean

DEFAULT

false

GROUP

skew_opt_variables

DESCRIPTION

By default, the skew_opt command does not consider clock gates during its optimization process. This means that critical clock gate enable slack issues are only resolved by requesting a latency reduction to the launching register of the enable path. Moreover, the impact of a register's latency reduction request on the upstream clock gates is not considered.

Setting the skew_opt_optimize_clock_gates variable to true enables the optimization of clock gates. It allows the skew_opt command to increase the latency to a clock gate to increase its enable slack. During the optimization process, the skew_opt command assumes that the latency between a clock gate and its downstream flops or clock gates cannot be reduced. In doing so, the skew_opt command understands the impact of a clock gate's latency increase on the downstream clock gates and flops. Moreover, the skew_opt command also understand the impact of a register's latency reduction on upstream clock gates.

Use the following command to determine the current value of the variable:

```
prompt> printvar skew_opt_optimize_clock_gates
```

SEE ALSO

skew_opt(2)
clock_opt(2)

skew_opt_skip_clock_balancing

Controls whether the **skew_opt** Tcl solution file applies **set_inter_clock_delay_options** commands when sourced.

TYPE

Boolean

DEFAULT

false

GROUP

skew_opt_variables

DESCRIPTION

The **skew_opt** command makes relative changes in clock network latencies (skews) to increase timing slack in a design. The changes are written to an output file, which is referred to as the "solution file". This file contains three types of settings:

```
Command group                  Controlling variable
-----------------------------  -------------------------------
set_clock_latency              skew_opt_skip_ideal_clocks
set_clock_tree_exceptions      skew_opt_skip_propagated_clocks
set_inter_clock_delay_options  skew_opt_skip_clock_balancing
```

Each type of command is grouped together and is applied only when the associated controlling variable is either **false** or undefined. Note that the controlling variables affect only which pieces of the solution file are active, not the behavior of the **skew_opt** command itself.

To determine the current value of the **skew_opt** variables, type **printvar skew_opt***.

SEE ALSO

`skew_opt(2)`

skew_opt_skip_ideal_clocks

Controls whether the **skew_opt** Tcl solution file applies **set_clock_latency** commands when sourced.

TYPE

Boolean

DEFAULT

false

GROUP

skew_opt_variables

DESCRIPTION

The **skew_opt** command makes relative changes in clock network latencies (skews) to increase timing slack in a design. The changes are written to an output file, which is referred to as the "solution file." This file contains three types of settings:

```
Command group                  Controlling variable
-----------------------------  -------------------------------
set_clock_latency              skew_opt_skip_ideal_clocks
set_clock_tree_exceptions      skew_opt_skip_propagated_clocks
set_inter_clock_delay_options  skew_opt_skip_clock_balancing
```

Each type of command is grouped together and is applied only when the associated controlling variable is either **false** or undefined. Note that the controlling variables affect only which pieces of the solution file are active, not the behavior of the **skew_opt** command itself.

To determine the current value of the **skew_opt** variables, type **printvar skew_opt***.

SEE ALSO

`skew_opt(2)`

skew_opt_skip_propagated_clocks

Controls whether the **skew_opt** Tcl solution file applies **set_clock_tree_exceptions** commands when sourced.

TYPE

Boolean

DEFAULT

false

GROUP

skew_opt_variables

DESCRIPTION

The **skew_opt** command makes relative changes in clock network latencies (skews) to increase timing slack in a design. The changes are written to an output file, which is referred to as the "solution file." This file contains three types of settings:

```
Command group                  Controlling variable
-----------------------------  -------------------------------
set_clock_latency              skew_opt_skip_ideal_clocks
set_clock_tree_exceptions      skew_opt_skip_propagated_clocks
set_inter_clock_delay_options  skew_opt_skip_clock_balancing
```

Each type of command is grouped together and is applied only when the associated controlling variable is either **false** or undefined. Note that the controlling variables affect only which pieces of the solution file are active, not the behavior of the **skew_opt** command itself.

To determine the current value of the **skew_opt** variables, type **printvar skew_opt***.

SEE ALSO

`skew_opt(2)`

skip_local_link_library

Specifies whether the local link library is used while linking.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether the local link library, as defined by the **local_link_library** attribute on the design, is used while linking the design.

By default (**false**), the local link library is used while linking the design.

When **true**, the local link library is skipped while linking the design. Sometimes the local link library contains settings that are irrelevant and should not be used for linking. In these cases, set this variable to **true** to skip the attribute while linking.

SEE ALSO

`set_local_link_library(2)`

skip_tie_cell_legalization

skip legalization step at end of connect_tie_cell.

TYPE

Boolean

DEFAULT

false

GROUP

placement

DESCRIPTION

connect_tie_cell command inserts needed tie cells at best locations and then call an internal ECO legalization engine to legalize the design. Sometimes users may prefer to skip this internal legalization step and prefers to call legalizer command later manually. This variable enable users to do so. Default is false.

Use the following command to determine the current value of the variable:

```
prompt> printvar skip_tie_cell_legalization
```

SEE ALSO

```
legalize_placement(2)
```

slice_signal_pin_vertically

Specifies the direction in which to slice signal pin polygons during blockage, pin, and via extraction.

TYPE

Boolean

DEFAULT

false

GROUP

```
extract_blockage_pin_via_variables
create_macro_fram_variables
```

DESCRIPTION

This variable specifies the direction in which to slice pin polygons during blockage, pin, and via extraction with the **extract_blockage_pin_via** or **create_macro_fram** command.

When this variable is set to **false** (the default), the FRAM extraction commands slice the signal pin polygons horizontally to create rectangles in the FRAM view. To have the tool slice the polygons vertically instead, set this variable to **true**.

SEE ALSO

```
extract_blockage_pin_via(2)
create_macro_fram(2)
```

sort_outputs

Sorts output ports on the schematic by port name.

TYPE

string

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

This variable sorts output ports on the schematic by port name.

spg_enable_ascii_flow

When set to true, you can run the **place_opt -spg** command in an ASCII flows.

TYPE

Boolean

DEFAULT

false

GROUP

icc-qor

DESCRIPTION

In a typical Synopsys Physical Guidance (SPG) flow from Design Compiler Graphical to IC Compiler, you would use a binary design database (MW or DDC) created after running the **compile_ultra -spg** command in Design Compiler Graphical. This is followed by reading this binary design database in IC Compiler, and issuing the **place_opt -spg** command. This command restores the SPG information for the standard cells from the design database.

Issuing the command **place_opt -spg** will error out in an ASCII flow, where you start by reading Verilog or VHDL instead of a binary design database from Design Compiler Graphical.

Setting this variable to true will allow the **place_opt -spg** command to complete successfully in an ASCII flow.

In order to run the **place_opt -spg** command in an ASCII flow, you will have to read in the Verilog/VHDL files along with the DEF file containing the Synopsys Physical Guidance generated by Design Compiler.

To generate the DEF file from Design Compiler, or for more information on SPG flows refer to the Design Compiler User Guide.

The default value for this variable is false.

SEE ALSO

`place_opt(2)`

supply_net_attributes

Describes the attributes related to supply nets.

DESCRIPTION

This man page describes the attributes related to supply nets. These attributes are defined only in UPF mode.

You can use the **get_attribute** command to determine value of an attribute, and use the **report_attribute** command to get a report of all the attributes on a specified supply net. To see all supply net attributes, use the **list_attribute -class supply_net -application** command.

Supply Net Attributes

cell_id

Specifies Milkyway design ID in which a supply net object is located.

This attribute is read-only.

internal_supply_net

Specifies whether a supply net object is internal.

When the supply net is set to internal, it can be connected with switched macro internal PG pins, derive_pg_connection won't try to connect this internal supply net.

This attribute is writable. You can use **set_attribute** to set its value.

name

Specifies name of a supply net object.

This attribute is read-only.

full_name

Specifies full name of a supply net object.

This attribute is read-only.

object_class

Specifies object class name of a supply net object, which is **supply_net**.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

This attribute is read-only.

within_block_abstraction

Specifies whether the supply net is part of the block abstraction.

The data type of **within_block_abstraction** is boolean.

This attribute is read-only and cannot be modified.

within_ilm

Specifies whether the supply net is part of an ILM.

The data type of **within_ilm** is boolean.

This attribute is read-only and cannot be modified.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
```

supply_port_attributes

Describes the attributes related to supply ports.

DESCRIPTION

This man pages describes the attributes related to supply ports. The attributes are defined only in UPF mode.

You can use the **get_attribute** command to determine value of an attribute, and use the **report_attribute** command to get a report of all attributes on a specified supply port. To see all supply port attributes, use the **list_attribute -class supply_port -application** command.

Supply Port Attributes

cell_id

Specifies Milkyway design ID in which a supply port object is located.

This attribute is read-only.

direction

Specifies the direction of a supply port object. The value can be "in" or "out".

This attribute is read-only.

full_name

Specifies full name of a supply port object.

This attribute is read-only.

name

Specifies name of a supply port object.

This attribute is read-only.

object_class

Specifies object class name of a supply port object, which is **supply_port**.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

This attribute is read-only.

within_block_abstraction

Specifies whether the supply port is part of the block abstraction.

The data type of **within_block_abstraction** is Boolean.

This attribute is read-only and cannot be modified.

within_ilm

Specifies whether the supply port is part of an ILM.

The data type of **within_ilm** is Boolean.

This attribute is read-only and cannot be modified.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
```

suppress_errors

Specifies a list of error codes for which messages are to be suppressed during the current shell session.

TYPE

list

DEFAULT

""

GROUP

system_variables

DESCRIPTION

This variable specifies a list of error codes for which messages are to be suppressed during the current shell session. The default is set to no error messages being suppressed.

EXAMPLES

The following example demonstrates the use of the variable.

```
set suppress_errors [list OPT-1406 OPT-1407 CMD-024 UIO-9 UIO-78]
```

symbol_library

Specifies the symbol libraries to use during schematic generation.

TYPE

string

DEFAULT

your_library.sdb

GROUP

schematic_variables

DESCRIPTION

This variable specifies the symbol libraries to use during schematic generation. This variable is a list of symbol library names.

synlib_enable_analyze_dw_power

Record power info of ungrouped DesignWare design.

TYPE

Integer

DEFAULT

0

GROUP

synlib_variables

DESCRIPTION

This variable enables recording power related info of ungrouped DesignWare design. The recorded info will be used for **analyze_dw_power** command.

When set to the default value of 0, no recording will happen. The report does not show the estimated power for ungrouped DesignWare designs. When set to 1, recording will be enabled. the **analyze_dw_power** command will show slack and power info for both ungrouped and non-ungrouped DesignWare design.

SEE ALSO

synlib_preferred_ff_chains

Specifies the preferred flip-flop chains for mapping FF chains in DesignWare Library Clock Domain Crossing (CDC) components.

TYPE

list

DEFAULT

""

GROUP

synlib_variables

DESCRIPTION

Specifies a list of metastability-hardened multiple flip-flop cells (chains) that Design Compiler uses when mapping the flip-flop chains in DesignWare Library CDC components, such as DW_sync.

For Design Compiler to consider the flip-flop chain cells during mapping, the technology library cells should meet the following requirements:

1. The technology library should contain the single-bit version of the flip-flop that is used in the flip-flop chain. A special string flag, "ff_chains", should exist on the single flip-flop for Design Compiler to find the chained library cells in the technology library.
2. The technology library cells that contain flip-flop chains should contain the same type of flip-flop. That is, all the flip-flops in the flip-flop chain should be the same.
3. The flip-flop chains in the technology library should have special integer flags, "ff_chain_stage", to indicate how many flip-flops are contained in the flip-flop chain cell.

The technology library vendor that creates the flip-flop chain library cell can define these required attributes in the library source code as "user_defined" attributes.

The flip-flop chain library cell can contain any number of single flip-flops. Design Compiler maps the flip-flop chains in DesignWare Library CDC components to the appropriate and preferred flip-flop chain library cells.

For example, if an instance of a DesignWare Library CDC component is configured to use four levels of synchronization and the preferred flip-flop chain library cells have 2, 3, and 4 flip-flops, the priority order is (from high to low):

One 4-flip-flop chain

Two 2-flip-flop chains

Four single flip-flops

Note that there are fewer requirements for single flip-flop metastability-hardened cells to be mapped as specified by the **synlib_preferred_ffs** variable.

SEE ALSO

`synlib_preferred_ffs(3)`

synlib_preferred_ffs

Specifies a list of preferred flip-flop (FF) library cells that DC will map multi-stage synchronizer FFs to within DesignWare Library Clock Domain Crossing (CDC) components.

TYPE

list

DEFAULT

""

GROUP

synlib_variables

DESCRIPTION

Specifies a list of single flip-flop (FF), metastability-hardened library cells that DC will use when technology mapping synchronizer FF chains in DesignWare Library CDC components (for example DW_sync).

If there are technology FF library cells that are designed specifically for synchronizers (sometimes referred to as "metastability-hardened" cells), you can set them as the preferred sychronizer FF by adding them to the synlib_preferred_ffs list. When DC maps the synchronizer FF chains within DesignWare Library CDC components, it tries to map them to the preferred FF(s).

Note: After mapping, DC can still size the FF cell. So it is still possible for the FF synchronizer chain used within DesignWare Library CDC components to be mapped to a non-preferred FF library cell.

SEE ALSO

`synlib_preferred_ff_chains(3)`

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)`

synthesized_clocks

Notifies budgeting that the design contains fully synthesized clock trees.

TYPE

binary

DEFAULT

false

GROUP

budgeting_variables

DESCRIPTION

Notifies budgeting that the design contains fully synthesized clock trees. Budgeting generates virtual clocks that are used to express the latencies of timing startpoints and endpoints of the launch and capture paths on the budgeted blocks.

SEE ALSO

`allocate_fp_budgets(2)`

synthetic_library

Specifies a list of synthetic libraries to use when compiling.

TYPE

list

DEFAULT

""

GROUP

system_variables, synlib_variables

DESCRIPTION

This variable specifies a list of synthetic libraries to use when compiling.

The **synthetic_library** variable works much like the **target_library** variable does for technology libraries. This variable can be set to be a list of zero or more sldb files that you want to use in the **compile** or **replace_synthetic** commands. When synthetic operators or modules are processed in compile, the operators, bindings, modules, and implementations of the specified library or libraries are used. Synthetic libraries are processed in order. So, if two modules in different libraries have the same name, the module in the first listed library is used.

As with target technology libraries, it is sometimes necessary to include your synthetic library as part of the link_library set. This is especially important when you instantiate synthetic modules.

Because HDL files automatically insert synthetic operators in a netlist, it is important to have a synthetic library defined that supports these operators. For this reason, the standard Synopsys library **standard.sldb** is automatically inserted as the first entry in the **synthetic_library** variable list. Then if a synthetic library is specified to be an empty list, the insertion of **standard.sldb** provides default definitions.

There is no way to disable the standard synthetic library, but you can disable individual modules or implementations by using the **dont_use** command. To replace a particular

implementation, disable it with **dont_use**, and use a replacement from your own synthetic library.

SEE ALSO

`target_library(3)`

system_variables

SYNTAX

```
Boolean auto_link_disable = "false"
string auto_link_options = "-all"
Boolean change_names_dont_change_bus_members = "false"
string command_log_file = "./command.log"
string company = ""
string compatibility_version = ""
Boolean context_check_status = true/false
string current_design = "<<undefined>>"
string current_instance =  "<<undefined>>"
string default_name_rules = ""
Boolean echo_include_commands = "true"
Boolean enable_page_mode = "true"
string exit_delete_filename_log_file = "true"
string filename_log_file = "filenames.log"
string link_force_case = "check_reference"
list link_library = {your_library.db}
list search_path = {. synopsys_root + "/libraries"}
list suppress_errors = {}
Boolean syntax_check_status = true/false
list synthetic_library = {}
list target_library = {your_library.db}
string uniquify_naming_style = "%s_%d"
Boolean verbose_messages = "true"
```

DESCRIPTION

These variables directly affect Synopsys Design Compiler and IC Compiler commands.

To view an individual variable description use the **man var** command where *var* is the variable name.

auto_link_disable

when **true** disables automatic linking of the design. Several Design Compiler and DFT Compiler commands (including **create_schematic** and **compile**) issue the **link** command automatically. During the execution of scripts containing thousands of **set_load** and **set_resistance** commands the overhead of automatic linking can be significant. This variable can be used in this situation to speed up the execution of such scripts. The default is **false**.

auto_link_options

Specifies the **link** command options to be used when **link** is invoked automatically by various Design Compiler and DFT Compiler commands (for example **create_schematic** and compile). The default is **-all**. To find the available options refer to the **link** command manual page.

change_names_dont_change_bus_members

This variable is for the **change_names** command. It affects bus members only of bussed ports or nets. When **false** (the default) **change_names** gives bus members the base name from their owning bus. For example if BUS A has range 0 to 1 with the first element NET1 and the second element NET2 **change_names** changes NET1 to A[0] and NET2 to A[1]. When this variable is set to **true change_names** does not change the names of bus members so that NET1 and NET2 remain unchanged.

command_log_file

Names the file to which a log of the initial values of variables and commands executed is to be written. If the value is an empty string a command log file is not created.

company

Names the company where Synopsys software is installed. The company name is displayed on the schematics.

compatibility_version

The name of the Synopsys software version to which the default behavior of the system should be set. This provides compatibility for script command files written in previous software versions. The scripts actually are run on the current version of the software so results are usually better. However the script performs the same default actions as it did on the specified software version.

context_check_status

One of a pair of status variables **syntax_check_status** and **context_check_status** whose values are set by the Syntax Checker and not by the user. A value of **true** indicates that the context_check mode is currently enabled; a value of **false** indicates that the mode is disabled.

current_design

The name of the design to be worked on. This variable is used by most of the Synopsys commands. Until a design is read into the system as the **current_design** the value of **current_design** is "<<undefined>>". When one or more designs are read into the system any of them can be made current by assigning the design name to this variable.

current_instance

The name of the instance to be worked on. Until a design is read into the system as the **current_design** the value of the **current_instance** is "<<undefined>>". When a design is

read into the system at the top hierarchical level the value of **current_instance** is the top design name followed by a slash.

default_name_rules

Contains the name of a name_rules file to be used as a default by **change_names** if a name rules file is not specified using the **-rules** option. The **change_names** command changes the names of ports, cells, and nets to conform to the rules contained in the file specified by the **default_name_rules** variable. For information about the format and creation of a name rules file, see the **change_names** man page.

echo_include_commands

When **true** (the default value), the **source** command prints the contents of files that it executes. When **false** contents are not printed.

enable_page_mode

When **true** (the default value), long reports are displayed one page at a time (similar to the UNIX **more** command).

exit_delete_filename_log_file

When **true** (the default value), causes the file specified by the **filename_log_file** variable to be deleted after the tool exits normally. Set the **exit_delete_filename_log_file** variable to **false** if you want the file to be retained.

filename_log_file

Specifies the name of the filename log file to be used in case a fatal error occurs during the tool execution. The file specified by the **filename_log_file** variable will contain the names of all files read in by the tool in the current session. If a fatal error occurs you can easily identify the data files needed to reproduce the fatal error. If this variable is not specified, the default filename files.log is used. This file is deleted if the program exits normally unless the **exit_delete_filename_log_file** variable is set to **false**.

link_force_case

Controls the case-sensitive or case-insensitive behavior of the **link** command. Values are **case_sensitive**, **case_insensitive**, or **check_reference**. The default is **check_reference**, which causes the **link** command to check and enforce the case sensitivity of the input format that created the reference.

link_library

Specifies the list of design files and libraries used during linking. The **link** command looks at those files and tries to resolve references in the order of specified files. If file names do not include directory names, files are searched for in the directories specified in the **search_path** variable. The default is {your_library.db}. You should change this to reflect your library name.

search_path

Specifies directories searched by the the tool for files specified without directory names. This variable is a list of directory names and is usually set to a central library directory.

suppress_errors

Specifies a list of error codes for which messages are to be suppressed during the current session.i By default, no error messages are suppressed.

syntax_check_status

One of a pair of status variables **syntax_check_status** and **context_check_status** whose values are set by the Syntax Checker and not by the user. A value of **true** indicates that the syntax_check mode is currently enabled; a value of **false** indicates that the mode is disabled.

synthetic_library

Specifies a list of synthetic libraries to use when compiling. The default is {}.

target_library

Specifies the list of technology libraries of components used when compiling a design. The default is {your_library.db}. You should change this to reflect your library name.

uniquify_naming_style

Specifies the naming convention used by the **uniquify** command. The variable string must contain only one %s (percent s) and %d (percent d) character sequence. To use a percent sign in the design name two are needed in the string (%%).

verbose_messages

When **true** causes more explicit system messages to be displayed during the current session. The default is **true**.

SEE ALSO

```
icc_shell(1)
change_names(2)
current_instance(2)
current_design(2)
if(2)
link(2)
syntax_check(2)
uniquify(2)
while(2)
```

target_library

Specifies the list of technology libraries of components to be used when compiling a design.

TYPE

list

DEFAULT

your_library.db

GROUP

system_variables

DESCRIPTION

This variable specifies the list of technology libraries containing the components to be used when compiling a design. The default value is *your_library.db*. Change this value to reflect your library name.

SEE ALSO

`system_variables(3)`

template_naming_style

Generates automatically a unique name when a module is built.

TYPE

string

DEFAULT

%s_%p

GROUP

hdl_variables

DESCRIPTION

This variable automatically generates a unique name when a module is built. This variable is one of three string variables that determine the naming conventions for parameterized modules (templates) built into a design through the **elaborate** command or automatically instantiated from an HDL file. The unique name automatically generated uses the module name, parameter names, and parameter values.

The **template_naming_style** variable determines what character or characters appear between the design name and the parameter name. The string value must contain %s, which represents the name of the original design, and %p, which represents the name and value of the parameter or parameters. You can optionally include any ASCII character or characters, or none, between %s and %p. For example, for a design named *DesignName* that has a parameter *parm1*, the default %s_%p causes the name **DesignName_parm1** to be generated.

Further, %s$%p, %s_*_%p, and %s%p would generate respectively the names **DesignName$parm1**, **DesignName_*_parm1**, and **DesignNameparm1**.

If a design has a noninteger parameter (or if **template_naming_style = ""**), the following definitions are locked down for these variables:

```
template_naming_style = %s_%p
template_parameter_style = %d
template_separator_style = _
```

SEE ALSO

```
template_parameter_style(3)
template_separator_style(3)
```

template_parameter_style

Generates automatically a unique name when a module is built.

TYPE

string

DEFAULT

%s%d

GROUP

hdl_variables

DESCRIPTION

This variable automatically generates a unique name when a module is built. This variable is one of three string variables that determine the naming conventions for parameterized modules (templates) built into a design through the **elaborate** command or automatically instantiated from an HDL file. The unique name automatically generated uses the module name, parameter names, and parameter values.

The **template_parameter_style** variable determines what character or characters appear between the parameter name and its value.

The string must contain %s, which represents the name of the parameter, and %d, which represents the value of the parameter. You can optionally include any ASCII character or characters, or none, between %s and %d. For example, for a parameter named *parm* that has a value of *1*, the default %s%d causes the name **parm1** to be generated.

The following are additional examples:

```
template_naming_style = "%s$%p"
template_parameter_style = %s_%d    results in DesignName$parm_1

template_naming_style = "%s_%p"
template_parameter_style = %s@%d    results in DesignName_parm@1
```

If a design has a noninteger parameter (or if **template_naming_style = ""**), the following definitions are locked down for these variables:

```
template_naming_style = %s_%p
template_parameter_style = %d
template_separator_style = _
```

SEE ALSO

template_naming_style(3)
template_separator_style(3)

template_separator_style

Generates automatically a unique name when a module is built.

TYPE

string

DEFAULT

_

GROUP

hdl_variables

DESCRIPTION

This variable automatically generates a unique name when a module is built. This variable is one of three string variables that determine the naming conventions for parameterized modules (templates) built into a design through the **elaborate** command or automatically instantiated from an HDL file. The unique name automatically generated uses the module name, parameter names, and parameter values.

The **template_separator_style** variable determines what character or characters appear between parameter names for templates that have more than one parameter. You can designate any ASCII character or characters, or none. The default value is an underscore (_).

For example, for a design called *DesignName* that has parameters named *parm1*, *parm2*, and *parm3*, if **template_naming_style = "%s_%p"** (the default value), and **template_separator_style = "_"**, the name **DesignName_parm1_parm2_parm3** is generated.

The following are additional examples:

```
template_naming_style = "%s$%p"
template_separator_style = "_"     results in
DesignName$parm1_parm2_parm3
template_naming_style = "%s#%p"
template_separator_style = "/"     results in DesignName#parm1/parm2/
parm3
```

If a design has a noninteger parameter (or if **template_naming_style = ""**), the following definitions are locked down for these variables:

```
template_naming_style = %s_%p
template_parameter_style = %d
template_separator_style = _
```

SEE ALSO

template_naming_style(3)
template_parameter_style(3)

terminal_attributes

DESCRIPTION

This man page described the attributes related to terminals (physical pins).

You can use the **get_attribute** command to determine value of an attribute, and use the **report_attribute** command to get a report of all the attributes on a specified object. To see all terminal attributes, use the **list_attribute -class terminal -application** command.

Note that attributes on a diode terminal are read-only.

Terminal Attributes

access_direction

Specifies allowable access directions for a terminal object.

The data type of **access_direction** is string.

Its valid values are:

- **right**
- **left**
- **up**
- **down**
- **unknown**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

Note that this attribute is undefined on diode terminal. You will get warning message with message ID "ATTR-3" when you try to get its value on diode terminal.

bbox

Specifies the bounding-box of a terminal. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ll

Specifies the lower-left corner of the bounding-box of a terminal.

The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **attr_name** of a terminal, by accessing the first element of its **bbox**.

The data type of **bbox_ll** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a terminal.

The data type of **bbox_llx** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a terminal.

The data type of **bbox_lly** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ur

Specifies the upper-right corner of the bounding-box of a terminal.

The \fbbox_ur is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ur** of a terminal, by accessing the second element of its **bbox**.

The data type of **bbox_ur** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a terminal.

The data type of **bbox_urx** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a terminal.

The data type of **bbox_ury** is double.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

cell_id

Specifies Milkyway design ID in which a terminal object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

constrained_status

Specifies constrained status of a terminal object.

The data type of **constrained_status** is string.

Its valid values are:

manual_created

side

location

order

unknown

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

Note that this attribute is undefined on diode terminal. You will get warning message with message ID "ATTR-3" when you try to get its value on diode terminal.

direction

Specifies direction of a terminal object.

The data type of **direction** is string.

Its valid values are:

in

out

inout

tristate

unknown

input

output

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

Note that the direction of a diode terminal will be marked as **diode**. You can not change its value by **set_attribute**.

double_pattern_mask_constraint

Specifies the coloring information of the terminal. This information is needed in the double patterning flow.

The data type of **double_pattern_mask_constraint** is string.

Its valid values are:

any_mask

mask1_soft

mask1_hard

mask2_soft

mask2_hard

same_mask

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

eeq_class

Specifies electrically equivalent class of a terminal object.

The data type of **eeq_class** is integer.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

Note that this attribute is undefined on diode terminal. You will get warning message with message ID "ATTR-3" when you try to get its value on diode terminal.

is_diode

Indicates whether a terminal object is diode.

The data type of **is_diode** is boolean.

This attribute is read-only.

is_fixed

Specifies whether a terminal object is marked as fixed.

The data type of **is_fixed** is boolean.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

layer

Specifies layer name of a terminal object.

The data type of **layer** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

must_join_class

Specifies must-join class of a terminal.

The data type of **must_join_class** is integer.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

Note that this attribute is undefined on diode terminal. You will get warning message with message ID "ATTR-3" when you try to get its value on diode terminal.

multiple_pattern_mask_constraint

Specifies the coloring information of the terminal. This information is needed in the multiple patterning flow.

The data type of **multiple_pattern_mask_constraint** is string.

Its valid values are:

any_mask

mask1_soft

mask1_hard

mask2_soft

mask2_hard

mask3_soft

mask3_hard

same_mask

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name

Specifies name of a terminal object.

The data type of **name** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

number_of_points

Specifies the number of points to illustrate the boundary of a terminal object.

The data type of **number_of_points** is integer.

You can refer to the attribute **points**. The list length of **points** is the value of **number_of_points**.

This attribute is read-only.

object_class

Specifies object class name of a terminal, which is **terminal**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

The data type of **object_id** is integer.

This attribute is read-only.

owner

Specifies Milkyway design file name in which a terminal is located.

The data type of **owner** is string.

This attribute is read-only.

owner_port

Specifies port name which a terminal object is associated with.

The data type of **owner_port** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

points

Specifies points of the boundary of a terminal. A terminal can be a rectangle, a rectilinear polygon, or multiple rectangles.

When a terminal is either a rectangle or a rectilinear polygon, its **points** is represented by a list of points. The last element of the list is the same as the first element.

When a terminal consists of multiple rectangles, its **points** is represented by a list of points of rectangles. Every five points represent one rectangle.

The data type of **points** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

side_status

Specifies constrained side name.

The data type of **side_status** is string.

Its valid values are:

left

right

bottom

top

unknown

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

Note that this attribute is undefined on diode terminal. You will get warning message with message ID "ATTR-3" when you try to get its value on diode terminal.

status

Specifies the placement status of a terminal object.

The data type of **status** is string. The valid values are:

fixed

placed

cover

unplaced

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

Note that this attribute is undefined on diode terminal. You will get an ATTR-3 warning message when you try to get its value on diode terminal.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

test_ate_sync_cycles

Specifies the number of ATE clock synchronization cycles used by a user-defined on-chip clocking (OCC) controller.

TYPE

positive integer

DEFAULT

1

GROUP

test_variables

DESCRIPTION

When you are using a user-defined on-chip clocking (OCC) controller, this variable specifies the number of ATE clock synchronization cycles used by the OCC controller when a scan-enable transition initiates a transition between the ATE and OCC clocks.

This variable affects the content of the protocol file written at the end of the **insert_dft** command.

By default, this variable is set to 1, which models an OCC controller that waits for a single clock cycle after a scan-enable transition. When scan-enable is reasserted after capture, scan data is delayed by one cycle, and the load_unload procedure is created as follows:

```
load_unload {
  V { SE=1; ATE_CLK=P; _si-=X; _so=X}
  Shift { ...}
}
```

You should change the value of this variable if your OCC controller does not need a synchronization cycle or if it requires more than one synchronization cycle after a scan-enable transition.

For more information on the synchronization logic structures and timing relationships, see SolvNet article 035708, "What Does the test_ate_sync_cycles Variable Do?".

Normally, this variable only affects the test protocol. If the variable is not set to the proper value, the test protocol can be edited or regenerated. However, in serialized compressed scan flows, it also affects the architecture of the serializer clock controller. In these flows, if the variable is not set to the proper value, the serializer logic must be reinserted with the variable set to the correct value.

SEE ALSO

test_bsd_input_ac_parametrics

Specifies what input port value transitions are tested in DC parametric patterns.

TYPE

Boolean

DEFAULT

false

GROUP

bsd_variables

DESCRIPTION

By default, the DC parametric patterns created by the **create_test_patterns -type {dc_parametric}** command exercise all port values for leakage characterization, but they do not exercise all value transitions for switching characterization.

With the default of **false**, input ports and bidirectional ports in input mode go through the following value sequence:

```
Input ports:
  X -> 0
  0 -> 1

Bidirectional ports as inputs:
  Z -> 0
  0 -> 1
```

By setting this variable to **true**, input ports and bidirectional ports input mode go through the following value sequence:

```
Input ports:
  X -> 0
  0 -> 1
  1 -> 0

Bidirectional ports as inputs:
  Z -> 0
```

```
0 -> 1
1 -> 0
```

SEE ALSO

`test_bsd_new_output_parametrics(3)`

test_bsd_new_output_parametrics

Specifies what output port value transitions are tested in DC parametric patterns.

TYPE

Boolean

DEFAULT

false

GROUP

bsd_variables

DESCRIPTION

By default, the DC parametric patterns created by the **create_test_patterns -type {dc_parametric}** command exercise all port values for leakage characterization, but they do not exercise all value transitions for switching characterization.

With the default of **false**, two-state output ports, three-state output ports, and bidirectional ports in output mode go through the following value sequence:

```
Two-state output ports:
  X -> 0
  0 -> 1

Three-state output ports, bidirectional ports as outputs:
  Z -> 0
  0 -> 1
```

By setting this variable to **true**, two-state output ports, three-state output ports, and bidirectional ports in output mode go through the following value sequence:

```
Output ports:
  X -> 0
  0 -> 1
  1 -> 0
  X -> 1

Three-state output ports, bidirectional ports as outputs:
```

```
Z -> 0
0 -> 1
1 -> 0
Z -> 1
```

SEE ALSO

`test_bsd_input_ac_parametrics(3)`

test_enable_codec_sharing

Instructs DFTMAX to share the same CODEC architecture across all partition decompressors/compressors.

TYPE

Boolean

DEFAULT

false

GROUP

test_variables

DESCRIPTION

When you insert adaptive scan compression logic with DFTMAX for multiple partitions at the top level, this variable controls whether a shared CODEC architecture is used across all partition decompressors/compressors (CODECs).

When set to its default value of **false**, smaller partition-specific CODECs are created for each partition. The top-level scan-in and scan-out ports are allocated separately to each partition CODEC, according to the total scan chain counts defined for the Internal_scan and ScanCompression_mode modes for each partition. Each scan-in and scan-out port connects to a single partition CODEC. There must be a sufficient number of scan-in and scan-out ports available to satisfy all partition CODEC connections.

When you set this variable to **true**, the scan chain counts are summed across the Internal_scan and ScanCompression_mode modes to build a more adaptive CODEC that is shared by all partitions. However, instead of building one large CODEC that connects to all partitions, this common CODEC is replicated for each partition. Functionally, all partitions share the same CODEC. Architecturally, this common CODEC is duplicated to keep routing local within the partition and to minimize top-level congestion.

For each duplicated CODEC, a separate subset of the decompressor outputs and compressor inputs are hooked up to each partition according to the ScanCompression_mode scan chain count for that partition. The scan-in ports are shared, so that each scan-in port drives all CODECs. The scan-out ports are combined using XOR

logic so that the top-level scan-out ports can capture the output from all partition compressors scan outputs. This architecture has similar testability efficiency to having a single top-level CODEC with the same architecture.

With the shared CODEC architecture, each partition's CODEC has only its subset of compressor and decompressor signals connected. After running **insert_dft**, you should perform an incremental compile with boundary scan optimization enabled to optimize the CODEC logic to each partition.

SEE ALSO

test_icg_n_ref_for_dft

Defines the integrated clock-gating cell to be used for implementing return-to-one clock gating in the DFT logic.

TYPE

string

DEFAULT

""

DESCRIPTION

When DFT Compiler inserts clock-gating cells as a part of the DFT logic, this variable constrains the insertion process to use the specified library cell to gate any clock signal that has a return-to-one (RTO) waveform. The name must match an available cell in the target libraries, and this cell must be an integrated clock-gating cell.

The cell specified by this variable is used only for the initial logic construction. After insertion, subsequent design optimizations can resize the cell (if allowed by the library cell attributes).

This variable applies to DFT logic inserted by DFTMAX during on-chip clocking (OCC) controller insertion and serializer insertion.

By default, when this variable is not set, DFTMAX will gate clock signals using regular combinational gates.

SEE ALSO

`test_icg_p_ref_for_dft(3)`

test_icg_p_ref_for_dft

Defines the integrated clock-gating cell to be used for implementing return-to-zero clock gating in the DFT logic.

TYPE

string

DEFAULT

""

DESCRIPTION

When DFT Compiler inserts clock-gating cells as a part of the DFT logic, this variable constrains the insertion process to use the specified library cell to gate any clock signal that has a return-to-zero (RTZ) waveform. The name must match an available cell in the target libraries, and this cell must be an integrated clock-gating cell.

The cell specified by this variable is used only for the initial logic construction. After insertion, subsequent design optimizations can resize the cell (if allowed by the library cell attributes).

This variable applies to DFT logic inserted by DFTMAX during on-chip clocking (OCC) controller insertion and serializer insertion.

By default, when this variable is not set, DFTMAX will gate clock signals using regular combinational gates.

SEE ALSO

`test_icg_n_ref_for_dft(3)`

test_occ_insert_clock_gating_cells

Enables the use of clock-gating cells in the implementation of the on-chip clocking (OCC) controller.

TYPE

Boolean

DEFAULT

false

GROUP

test_variables

DESCRIPTION

By default, when a DFT-inserted on-chip clocking (OCC) controller is implemented, combinational multiplexer logic is used to switch between the clocks. You can set this variable to **true** to use latch-based clock-gating clock selection logic instead.

When this variable is set to **true**, the following capabilities are also available:

- If you want to use a particular integrated clock-gating (ICG) cell, you can specify it with the **test_icg_p_ref_for_dft** variable.
- If you want to use particular library cells for the clock selection logic, you can specify them with the **occ_lib_cell_nor2**, **occ_lib_cell_andor21**, **occ_lib_cell_andor22**, **occ_lib_cell_clkbuf**, and **occ_lib_cell_clkinv** design attributes (in supported combinations).
- If you want a dedicated OCC clock chain connection that exists only on the ATE clock path, you can set the **test_dedicated_clock_chain_clock** variable to **true**.

SEE ALSO

`test_icg_p_ref_for_dft(3)`

test_serialize_put_fsm_clock_output

Allows the serializer clock controller outputs to be described in the STIL protocol file.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

By default, when DFTMAX inserts a serializer, it includes information about the outputs of the serializer clock controller in the STIL protocol file for ScanCompression_mode. The information is placed in a ClockStructure block. This information helps TetraMAX trace the source of the signal clock signals during DRC.

You can prevent the addition of this information by setting this variable to **false**.

SEE ALSO

text_attributes

Contains attributes related to text.

DESCRIPTION

Contains attributes related to text.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class text -application**, the definition of attributes can be listed.

Text Attributes

anchor

Specifies the anchor position for text from its **origin**.

The data type of **anchor** is string.

Its valid values are:

- **lb** - Left Bottom
- **cb** - Center Bottom
- **rb** - Right Bottom
- **lc** - Left Center
- **c** - Center Center
- **rc** - Right Center
- **lt** - Left Top
- **ct** - Center Top
- **rt** - Right Top

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

bbox

Specifies the bounding-box of a text. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding-box of a text.

The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **attr_name** of a text, by accessing the first element of its **bbox**.

The data type of **bbox_ll** is string.

This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a text.

The data type of **bbox_llx** is double.

This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a text.

The data type of **bbox_lly** is double.

This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a text.

The \fbbox_ur is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ur** of a text, by accessing the second element of its **bbox**.

The data type of **bbox_ur** is string.

This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a text.

The data type of **bbox_urx** is double.

This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a text.

The data type of **bbox_ury** is double.

This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a text object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

height

Specifies height of a text object.

The data type of **height** is float.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

layer

Specifies layer name of a text object.

The data type of **layer** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name

Specifies name of a text object.

The data type of **name** is string.

This attribute is read-only.

object_class

Specifies object class name of a text, which is **text**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

The data type of **object_id** is integer.

This attribute is read-only.

orientation

Specifies orientation of a text object.

The data type of **orientation** is string.

Its valid values are:

N

E

S

W

FN

FE

FS

FW

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

origin

Specifies origin of a text object.

The **origin** is represented by a **point**. The format of a *point* specification is {x y}.

The data type of **origin** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

text

Specifies the text string to create and display.

The data type of **text** is string.

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

text_editor_command

Specifies the command that executes when the **Edit/File** menu is selected in the Design Analyzer text window.

TYPE

string

DEFAULT

xterm -fn 8x13 -e vi %s &

GROUP

view_variables

DESCRIPTION

Specifies the command that executes when the **Edit/File** menu is selected in the Design Analyzer text window.

To determine the current value of this variable, type **printvar text_editor_command**. For a list of all **view** variables and their current values, use the **print_variable_group view** command.

SEE ALSO

text_print_command

Specifies the command that executes when the **File/Print** menu is selected in the Design Analyzer text window.

TYPE

string

DEFAULT

lpr -Plw

GROUP

view_variables

DESCRIPTION

Specifies the command that executes when the **File/Print** menu is selected in the Design Analyzer text window.

To determine the current value of this variable, use **printvar text_print_command**. For a list of all **view** variables and their current values, use the **print_variable_group view** command.

SEE ALSO

tieoff_hierarchy_opt

Controls whether to allow tie-off optimization down to the lowest hierarchy without port punching during fixing DRC violations of constant nets that cross multiple hierarchies.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

In RMgen reference methodology, tie-off optimization is called at the end of the **place_opt** or **psynopt** command to fix the DRC violations of constant nets that have maximum capacitance or maximum fanout.

Setting this variable to true allows tie-off optimization down to the lowest hierarchy without port punching to fix constant nets that cross multiple hierarchies.

For example, a violating constant net that drives an input port followed by multiple loads, the tool fixes the violation by inserting an optimized number of tie cells down to the lowest hierarchy to drive the loads and removing the original driving cell. You can choose to keep the original driving cell by setting the **tieoff_hierarchy_opt_keep_driver** variable.

This variable does not support multivoltage or multicorner-multimode designs. It also skips the constant nets that have the dont_touch attribute.

For multicorner-multimode designs, this variable uses information from the current scenario only.

SEE ALSO

```
tieoff_hierarchy_opt_keep_driver(3)
set_fix_multiple_port_nets(2)
```

tieoff_hierarchy_opt_keep_driver

Controls whether to keep the original driving cell when propagating tie-off optimization down to the lowest hierarchy.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When the **tieoff_hierarchy_opt** variable is set to true, tie-off optimization is allowed to propagate to the lowest hierarchy to fix constant nets that cross multiple hierarchies. By default, the original driving cell of the constant net is removed.

This variable is used to control whether to remove the original driving cell of the constant net. When you set it to true, the original driving cell is not removed.

This variable does not support multivoltage or multicorner-multimode designs. For multicorner-multimode designs, this variable uses the information from the current scenario only.

SEE ALSO

`tieoff_hierarchy_opt(3)`

timing_aocvm_analysis_mode

Configures advanced on-chip variation (AOCV) analysis.

TYPE

string

DEFAULT

combined_clock_and_data_metrics

GROUP

timing_variables

DESCRIPTION

This variable sets the configuration for advanced on-chip variation (AOCV) analysis. To perform AOCV analysis, the **timing_aocvm_enable_analysis** variable must be set to **true**.

In AOCV analysis, the derating factor for a path depends on the path depth, which is defined as the number of successive cell (or net) delay timing arcs in a path from the common point to the endpoint. The depth ranges and corresponding derating factors are specified in a table, which is read into the tool with the **read_aocvm** command. Separate depth values are calculated for cells and nets, and for launch and capture paths.

When this variable is set to **""**, the default AOCV analysis is performed. AOCV derating is applied throughout the design. Both clock and data network objects are included in the depth computation.

When this variable is set to **clock_network_only**, AOCV derating is applied to arc delays in the clock network only. Constant (OCV) derating is applied to data paths, if constant derating factors have been annotated for data objects; otherwise they are not derated. Depth is calculated based on clock network topology only; delay timing arcs in the data network are excluded from depth calculations.

When this variable is set to **separate_data_and_clock_metrics**, separate depths are computed for the clock and data paths and the appropriate AOCV derating factors based on the separate depths are used for derating clock and data arc delays.

When this variable is set to **combined_launch_capture_depth**, launch and capture depths are not calculated separately. The launch and capture paths are considered together and a combined depth is calculated for the entire path. This option cannot be used together with the **separate_data_and_clock_metrics** mode.

The **timing_aocvm_ocv_precedence_compatibility** variable specifies whether to consider on-chip variation (OCV) derating in addition to as AOCV derating while AOCV is enabled.

SEE ALSO

```
read_aocvm(2)
remove_aocvm(2)
report_aocvm(2)
timing_aocvm_enable_analysis(3)
timing_aocvm_ideal_clock_depth(3)
timing_aocvm_ideal_clock_mode(3)
timing_aocvm_ocv_precedence_compatibility(3)
```

timing_aocvm_enable_analysis

Enables advanced on-chip variation (AOCV) analysis.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

When this variable is set to **true**, the tool uses graph-based AOCV timing analysis, which applies different derating values to different arcs based on the depth of the path (the number of successive gates in the path).

The derating values must be specified in a table and read into the tool with the **read_aocvm** command.

When **false** (the default), the tool performs conventional timing derating. In this mode, constant timing derating values specified with the **set_timing_derate** command are required to pessimistically bound the analysis.

When the variable is set to **true**, the tool performs AOCV analysis. By default, advanced OCV analysis applies only to the clock network. The **timing_aocvm_analysis_mode** variable lets you choose to apply AOCV analysis to data paths as well as clock paths.

SEE ALSO

```
read_aocvm(2)
remove_aocvm(2)
report_aocvm(2)
timing_aocvm_analysis_mode(3)
```

timing_aocvm_enable_distance_analysis

Enables distance analysis support in advanced on-chip variation (AOCV) analysis.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

When this variable is set to true, the tool calculates derating values for timing arcs based on both the distance of the path (physical range of the path) and the depth of the path (the number of successive gates in the path).

When this variable is set to false, the tool uses only uses the worst-case distance information in the table to calculate the derating values, which is more pessimistic than using all of the distance information.

This variable setting has an effect only under the following conditions:

- Advanced on-chip variation analysis has been enabled by setting the **timing_aocvm_enable_analysis** variable to true.
- The AOCV data tables contain distance dependency information.
- The tool has access to information about the physical locations of the cells and nets in the timing paths, so it can calculate the physical span of each timing arcs.
- Setting this variable to false results in the behavior of older releases of the tool. You should set this variable before you read in any AOCV data tables. Otherwise, you need to use the **read_aocvm** commands to read the tables again.

SEE ALSO

```
read_aocvm(2)
remove_aocvm(2)
report_aocvm(2)
timing_aocvm_enable_analysis(3)
```

timing_aocvm_ideal_clock_depth

Specifies the clock network depth when the **timing_aocvm_ideal_clock_mode** variable is set to **true**.

TYPE

float

DEFAULT

0

GROUP

timing_variables

DESCRIPTION

This variable specifies the clock network depth (number of successive gates in each clock path) when the **timing_aocvm_ideal_clock_mode** variable is set to true.

In this advanced on-chip variation (AOCV) analysis mode, the launch and capture depths are not calculated from the clock network topology. Instead, you explicitly specify the path depth in the clock network using the **timing_aocvm_ideal_clock_depth** variable. This same depth value applies to all clock paths.

You can use this mode with ideal clocks, before clock tree synthesis, when information about the actual number of gates in the clock network is not yet known. Specify the approximate number gates you expect in the clock tree for typical clock paths.

SEE ALSO

```
timing_aocvm_enable_analysis(3)
timing_aocvm_ideal_clock_mode(3)
```

timing_aocvm_ideal_clock_mode

Enables ideal clock analysis support in advanced on-chip variation (AOCV) analysis.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

When this variable is set to **true**, the tool calculates the AOCV depths based on the user-specified clock network depth using the **timing_aocvm_ideal_clock_depth** variable.

When **false** (the default), the tool performs conventional AOCV analysis, in which depths are calculated from the actual network topology.

You can set this variable to **true** with ideal clocking, before clock tree synthesis, when information about the actual number of gates in the clock network is not yet known.

SEE ALSO

```
read_aocvm(2)
remove_aocvm(2)
report_aocvm(2)
timing_aocvm_enable_analysis(3)
timing_aocvm_ideal_clock_depth(3)
```

timing_aocvm_ocv_precedence_compatibility

Specifies whether to fall back to conventional on-chip variation (OCV) derating while advanced on-chip variation (AOCV) derating is enabled.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

When this variable is set to false (the default), for each object, both OCV and AOCV derate settings are considered, and the more specific setting is used. This is the default order of usage, from highest to lowest priority:

```
   i.  OCV leaf cell derate setting
  ii.  AOCV lib-cell derate setting
 iii.  OCV lib-cell derate setting
  iv.  AOCV hier-cell derate setting
   v.  OCV hier-cell derate setting
  vi.  AOCV design derate setting
 vii.  OCV design derate
```

When this variable is set to true, the OCV derate settings are always ignored for AOCV analysis, resulting in the following order of usage, from highest to lowest priority:

```
   i.  AOCV lib-cell derate setting
  ii.  AOCV hier-cell derate setting
 iii.  AOCV design derate setting
```

This variable has an effect only when the **timing_aocvm_enable_analysis** variable is set to true.

SEE ALSO

```
read_aocvm(2)
remove_aocvm(2)
report_aocvm(2)
timing_aocvm_enable_analysis(3)
```

timing_ccs_load_on_demand

Enables or disables incremental loading of Composite Current Source (CCS) libraries.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

This variable enables incremental (or on-demand) loading of CCS data for libraries. Because of the requirement to frequently load many libraries, cell libraries that contain CCS timing data can quickly consume all available memory. At read time, the library is analyzed so that the timer can later load CCS data for only the library cells that require it. There is a small upfront cost in runtime at the library loading phase in exchange for a significant saving of memory.

You must have some temporary disk space available in either /tmp or the directory pointed to by the TMPDIR environment variable. It is recommended that you have at least as much free space as the size of the original library files for the libraries that are to be read. All temporary files created for this purpose are deleted at program exit time. For best performance, temporary directories should be located on a disk that is mounted as a local filesystem.

Note that commands like **report_lib** and **write_lib**, which require CCS information for all library cells, will still operate correctly. However, while these commands are running, they might negate memory savings obtained by using the **timing_ccs_load_on_demand** variable for the library they are analyzing.

SEE ALSO

UI-200(n)

timing_check_defaults

Defines the default check list in the **check_timing** command.

TYPE

string

DEFAULT

generated_clock loops no_input_delay unconstrained_endpoints pulse_clock_cell_type no_driving_cell partial_input_delay

DESCRIPTION

This variable defines the default checks to be performed when the **check_timing** command is run without any options. The default check list defined by this variable can be overridden by redefining the check list. The check list modified using the **-override_defaults**, **-include**, or **-exclude** option of **check_timing** is valid in only one command.

Note that this variable will not check if the value is correct or not; the check is done by the **check_timing** command. Each element in the check list can be one of the following strings:

```
loops
no_input_delay
unconstrained_endpoints
generated_clock
pulse_clock_cell_type
clock_crossing
data_check_multiple_clock,
data_check_no_clock
multiple_clock
generic
gated_clock
ideal_timing
retain
clock_no_period
```

EXAMPLE

The following example defines the value of the **timing_check_defaults** variable:

```
prompt> set timing_check_defaults {clock_crossing loops}
```

SEE ALSO

check_timing(2)

timing_clock_gating_propagate_enable

Allows the gating-enable signal delay to propagate through the gating cell.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable, when set to **true**, allows the delay and slew from the data line of the gating check to propagate. When set to **false**, the tool blocks the delay and slew from the data line of the gating check from propagating; only the delay and slew from the clock line is propagated.

If the output goes to a clock pin of a latch, setting this variable to **false** produces the most desirable behavior.

If the output goes to a data pin, setting this variable to **true** produces the most desirable behavior.

SEE ALSO

timing_clock_reconvergence_pessimism

Selects signal transition sense matching for computing clock reconvergence pessimism removal.

TYPE

string

DEFAULT

normal

DESCRIPTION

This variable determines how the value of the clock reconvergence pessimism removal (CRPR) is computed with respect to transition sense. Allowed values are **normal** (the default) and **same_transition**.

When set to **normal**, the CRPR value is computed even if the clock transitions to the source and destination latches are in different directions on the common clock path. It is computed separately for rise and fall transitions and the value with the smaller absolute value is used.

When set to **same_transition**, the CRPR value is computed only when the clock transition to the source and destination latches have a common path and the transition is in the same direction on each pin of the common path. Thus, if the source and destination latches are triggered by different edge types, CRPR is computed at the last common pin at which the launch and capture edges match.

If the variable is set to **same_transition**, the CRPR for all minimum pulse width checks will be zero, as they are calculated using different (rise and fall) clock edges.

SEE ALSO

```
report_timing(2)
timing_crpr_remove_clock_to_data_crp(3)
timing_crpr_threshold_ps(3)
timing_remove_clock_reconvergence_pessimism(3)
```

timing_consider_internal_startpoints

Determines whether to report timing paths that start from internal startpoints.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** (the default), timing paths that start from internal startpoints are reported by the **report_timing** command. When this variable is set to **false**, timing analysis skips the internal startpoints, so these paths are not reported.

Internal startpoints can result from unresolved references, cells with disabled timing arcs, or input ports with only the rising-edge or only the falling-edge input delay specified with the **set_input_delay** command.

SEE ALSO

```
report_timing(2)
get_timing_paths(2)
set_input_delay(2)
```

timing_crpr_remove_clock_to_data_crp

Allows the removal of clock reconvergence pessimism (CRP) from paths that fan out directly from the clock source to the data pins of sequential devices.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, clock reconvergence pessimism (CRP) will be removed for all paths that fan out directly from the clock source pins to the data pins of sequential devices. All sequential devices that reside in the fanout of clock source pins must be handled separately in the subsequent timing update. This might cause a severe performance degradation to the timing update.

SEE ALSO

`timing_remove_clock_reconvergence_pessimism(3)`

timing_crpr_remove_muxed_clock_crp

Allows clock reconvergence pessimism removal (CRPR) to consider common path reconvergence between related clocks.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

This variable controls the clock reconvergence pessimism removal (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.

By default, 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. When 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 that switch dynamically, set this variable to **false** so the CRP is not removed. Related clocks switch dynamically when a timing path launches from one related clock and the clock steering logic switches dynamically so the path captures on the other related clock.

The default value is **true**, which removes the additional CRP.

SEE ALSO

```
timing_remove_clock_reconvergence_pessimism(3)
```

timing_crpr_threshold_ps

Specifies the 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 pico seconds (ps), regardless of the units of the main library.

The threshold is per reported slack: setting this variable to the value of *TH1* means that reported slack is no worse than *S* - *TH1*, where *S* is the reported slack when the **timing_crpr_threshold_ps** variable is set close to zero (the minimum allowed value is **2e-5** pico seconds).

The variable has no effect if CRPR is not active; the **timing_remove_clock_reconvergence_pessimism** variable is set to **false**). The larger the value of **timing_crpr_threshold_ps**, the faster the runtime when CRPR is active. The recommended setting is about half stage (gate plus net) delay of a typical gate in the clock network. It provides a reasonable trade-off between accuracy and runtime in most cases.

You might want to use different settings throughout the design cycle: larger during the design phase, smaller for sign-off. You might want to experiment and set a different value when moving to a different technology.

SEE ALSO

`timing_remove_clock_reconvergence_pessimism(3)`

timing_disable_cond_default_arcs

Disables the default, nonconditional timing arc between pins that do have conditional arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable disables nonconditional timing arcs between any pair of pins that have at least one conditional arc. By default, these nonconditional timing arcs are not disabled. This variable is primarily intended to deal with the situation between two pins that have conditional arcs, where there is always a default timing arc with no condition.

Set this variable to **true** when the specified conditions cover all possible state-dependent delays, so that the default arc is useless. For example, consider a 2-input XOR gate with inputs as A and B and with output as Z. If the delays between A and Z are specified with 2 arcs with respective conditions 'B' and 'B~'', the default arc between A and Z is useless and should be disabled.

SEE ALSO

`report_disable_timing(2)`

timing_disable_recovery_removal_checks

Controls whether the tool accepts recovery and removal arcs specified in the technology library.

TYPE

Boolean

DEFAULT

true

GROUP

compile_variables

DESCRIPTION

This variable, when set to **false**, enables the acceptance of recovery and removal arcs specified in the technology library. Recovery or removal timing arcs impose constraints on asynchronous pins of sequential cells. Typically, recovery time specifies the time the inactive edge of the asynchronous signal has to arrive before the closing edge of the clock. Removal time specifies the length of time the active phase of the asynchronous signal must be held after the closing edge of clock.

To enable the **compile**, **report_timing**, and **report_constraint** commands to accept recovery or removal arcs specified in the library, set **timing_disable_recovery_removal_checks** to **false**.

Note that independent of the value of this variable, the **write_timing** and **report_delay_calculation** commands always accept and report recovery or removal timing information.

This variable is the logical opposite of the variable **enable_recovery_removal_arcs**. If you set either one of these variables to **false**, the tool automatically sets the other variable to **true**, and vice versa.

SEE ALSO

`enable_recovery_removal_arcs(3)`

timing_dont_traverse_pg_net

Normally, timing analysis will be processed only upon signal network and ignore Power/Ground pins. However, command **derive_pg_connection** might bring some netlist changes and make PG nets being connected to normal signal network.

In that case, timing analysis will pay huge runtime for traverse in PG networks and then bring some runtime issues. Also, case-analysis needs to honor the logic-constants bounding to PG Power/Ground.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

This variable is used to configures whether timing analysis should touch PG networks or not. By default, timing analysis will touch PG networks and this could bring some runtime issues.

When this variable is set to **true**, timing analysis will skip traverse in PG networks and then save runtime. Meanwhile, case-analysis will honor the logic-constants pins/ports which have been bounded to Power/Ground separately.

When this variable is set to **false**, timing analysis will do traverse in PG networks, along with normal signal networks. The logic-constants pins/ports which have been bounded to Power/Ground will be ignored by case-analysis as well.

The default variable value is **false**.

SEE ALSO

`set_case_analysis(2)`

```
report_timing(2)
```

timing_early_launch_at_borrowing_latches

Removes clock latency pessimism from the launch times for paths that begin at the data pin of a transparent latch.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

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 situation, whenever the IC Compiler tool determines that time borrowing is occurring at such a D-pin, it creates a timing path starting at the D-pin.

Sometimes there is a difference between the launch and capture latch latencies, due either to reconvergent paths in the clock network or different min and max delays of cells in the clock network. For setup paths, the tool uses the late value to launch and the early value to capture. This achieves the tightest constraint and prevents optimism. However, for paths starting from latch D-pins, this analysis is pessimistic because the data simply passes through and thus does not even "see" the clock edge at the latch.

When this timing variable is set to **true** (the default), such pessimism is eliminated by using the early latch latency to launch the path. Note that only paths that originate from a latch D-pin are affected. When the variable is set to **false**, late clock latency is used to launch all setup paths in the design.

When the **timing_early_launch_at_borrowing_latches** and **timing_remove_clock_reconvergence_pessimism** variables are both set to true, it is not possible to apply both pessimism removal techniques on the same timing path. Therefore, the tool applies early launch pessimism removal to paths that start from a transparent latch D-pin and applies CRPR to the remaining paths.

For a design with long clock paths, it might be preferable to set the **timing_early_launch_at_borrowing_latches** variable to **false**. Doing so allows CRPR to be applied to all paths, including those that start from a transparent latch D-pin. When clock paths are long, CRPR can be the more powerful pessimism reduction technique.

On the other hand, for a design with clock paths having long common segments, or where critical paths traverse several latches, leaving the **timing_early_launch_at_borrowing_latches** variable set to **true** might result in more pessimism removal overall, even though paths that start from a transparent D-pin do not get any CRPR credit.

SEE ALSO

```
report_timing(2)
timing_remove_clock_reconvergence_pessimism(3)
```

timing_edge_specific_source_latency

Controls whether or not the generated clock source latency computation considers edge relationship.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to **true**, only the paths with the same sense relationship derived from generated clock definition are considered.

By default, all paths that fanout to a generated clock source pin are considered and the worst path is selected for generated clock source latency computation.

SEE ALSO

`create_generated_clock(2)`

timing_enable_multiple_clocks_per_reg

Enables analysis of multiple clocks that reach a single register.

TYPE

Boolean

DEFAULT

true

GROUP

timing_variables

DESCRIPTION

This variable enables analysis of multiple clocks that reach a register clock pin. When set to **true** (the default), all clocks reaching the register are analyzed simultaneously, including interactions between different clocks, such as data launch by one clock and data capture by another. If there are four or more clocks per register and the design contains level-sensitive registers, a high impact on runtime might occur.

If your design has a large number of different interacting clocks, you can set this variable to **false** to eliminate consideration of all clock interactions and get results more quickly. However, to get fully accurate results, leave the variable set to **true** and use the **set_false_path** command to explicitly specify the actual false interactions between mutually exclusive clocks.

SEE ALSO

```
check_timing(2)
create_clock(2)
create_generated_clock(2)
set_false_path(2)
```

timing_enable_non_sequential_checks

Enables or disables library nonsequential checks in the design.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables or disables analysis of library nonsequential checks in the design. The nonsequential arcs defined in the library will not be used for constraint checking unless this variable is set to **true**. This variable does not affect the data checks defined by the **set_data_check** command.

Enabling the nonsequential checks might cause delays if the signals reaching the related pin and the constrained pin do not belong to the same clock domain. Use the **set_multicycle_path** command to put appropriate constraints on such paths.

SEE ALSO

```
printvar(2)
set_data_check(2)
set_multicycle_path(2)
```

timing_enable_normalized_slack

Enables normalized slack analysis during timing updates.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When set to **true**, this variable enables normalized slack analysis during timing updates. When set to **false** (the default), this variable disables normalized slack analysis and reporting.

Normalized slack analysis is an optional analysis method that calculates normalized slack for each timing path:

```
normalized_slack = path_slack /
idealized_allowed_propagation_delay_for_path
```

The tool computes the allowed propagation delay for the path using ideal clock edges; it ignores setup time, uncertainty, and clock latency. The allowed propagation delay can be a half-cycle, a full cycle, or multiple cycles. It can be more complicated to compute when the launch and capture clocks are different.

Normalized slack analysis can be used to determine the paths that limit the clock frequency. A normalized slack report prioritizes violating paths that are allowed few clock cycles. Fixing these paths first results in the most improvement in the clock period.

If normalized slack analysis is enabled during update timing, you can gather and report paths according to normalized slack by using the commands **report_timing -normalized_slack** and **get_timing_paths -normalized_slack**.

SEE ALSO

```
printvar(2)
report_timing(2)
```

```
timing_max_normalization_cycles(3)
```

timing_enable_preset_clear_arcs

Controls whether timing analysis enables or disables preset and clear arcs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable, when set to **true**, permanently enables asynchronous preset and clear timing arcs, so that you can use them to analyze timing paths. When set to **false** (the default), timing analysis disables all preset and clear timing arcs.

If there are any minimum pulse width checks defined on asynchronous preset and clear pins, they are performed regardless of the value of this variable. Also, the **-true** and **-justify** options of the **report_timing** command cannot be used unless this variable is at its default value.

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

SEE ALSO

```
printvar(2)
report_timing(2)
```

timing_enable_slack_distribution

Determines whether to distribute slack evenly along time-borrowing paths.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

This variable determines whether to distribute slack evenly along a time-borrowing path containing transparent latches.

By default, this variable is set to **false**, resulting in "greedy" time-borrowing behavior, in which latch time borrowing matches the D pin arrival time, producing zero slack at borrowing latches.

When this variable is set to **true**, during optimization, the tool sets the amount of time borrowing at transparent latches in a manner that distributes negative or positive slack evenly among the latches of a multistage latch path. The runtime and memory usage might be longer using this setting.

Slack distribution spreads all nonzero slack, negative or positive, across all multistage borrowing latch paths. By spreading positive path slacks, hold fixing and area QoR might be improved. This setting affects the behavior of the **place_opt** and **route_opt** commands.

Slack distribution, when enabled, overrides automatic time borrowing (controlled by the **disable_auto_time_borrow** variable). Automatic time borrowing distributes only the worst negative slack across certain multistage latch paths.

SEE ALSO

```
set_max_time_borrow(2)
disable_auto_time_borrow(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), this variable disables advanced analysis and reporting through transparent latches.

By default timer 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.

With the advanced analysis through transparent latches, paths through latches can be reported as a single timing path. Pin slacks can be affected by timing calculations past the first level of latches in the fanout. In addition, specific paths through latches can be requested 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 variable **timing_enable_through_paths** 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. The user 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**. The variable **timing_through_path_max_segments** can be used to control the selection of these pins.

To determine the current setting, use the following command:

```
printvar timing_enable_through_paths
```

SEE ALSO

```
printvar(2)
report_timing(2)
timing_through_path_max_segments(3)
```

timing_gclock_source_network_num_master_registers

Specifies the maximum allowed number of register clock pins clocked by the master clock in generated clock source latency paths.

TYPE

integer

DEFAULT

10000000

DESCRIPTION

This variable controls the maximum allowed number of register clock pins clocked by the master clock in generated clock source latency paths. The variable does not limit the number of registers 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 with a primary master clock that differs from the generated clock whose source latency paths are being computed.

timing_ignore_paths_within_block_abstraction

Causes timing paths entirely within a block abstraction to be ignored.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The process of creating abstract blocks removes as much logic as possible from the internal portions of the block, while keeping cells and nets that are necessary to preserve paths that connect to the outside of the block. However, in a block abstraction, there can be residual paths entirely within the boundary logic that do not traverse outside the block.

Setting this variable to **true** causes the timing on paths that remain entirely within the abstracted block to be ignored. The tool behaves as if there is an implied false path constraint on such paths, both for reporting and for optimization. A path that starts and ends inside and abstracted block, but exits from and returns to the block somewhere in the middle, is not affected by this variable.

When this variable is **true**, it allows transparent interface optimization to focus on paths that traverse outside the abstracted blocks.

SEE ALSO

`create_block_abstraction(2)`

timing_input_port_default_clock

Determines whether a default clock is assumed at input ports for which a clock-specific input external delay is not defined.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

This variable affects the behavior of the synthesis timing engine when timing a path from an input port with no clocked input external delay. When set to **true**, all such input ports are given 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. By default, no such imaginary clock is assumed.

SEE ALSO

`report_timing(2)`

timing_library_derate_is_scenario_specific

This variable determines whether the **set_timing_derate** and the **read_aocvm** commands apply to the current scenario or all scenarios for library cell objects.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

By default, the **set_timing_derate** command and the **read_aocvm** command, when applied to a library cell object, affect the timing of that cell in all scenarios, not just the scenario in which the command is executed. If you set this variable to true, the **set_timing_derate** and the **read_aocvm** commands affect only the scenario in which the commands are executed. The variable is set to false by default.

The variable setting has no effect on the **set_timing_derate** and **read_aocvm** commands applied to instances, in which case the **set_timing_derate** setting and the AOCVM tables are always scenario-specific.

Changing this variable at any time removes all timing derate settings and AOCVM tables already applied to library cell objects. To save runtime, set this variable early in the session, before you start using the **set_timing_derate** and **read_aocvm** commands.

This variable is not persistent and is not saved in the Milkyway design database.

Multicorner-Multimode Support

This variable determines whether the **set_timing_derate** command and the **read_aocvm** command apply to the current scenario or all scenarios for settings applied to library cell objects.

EXAMPLE

The following example shows the default behavior. Here, for the **set_timing_derate** command, the previous setting "1.1" is overwritten by the later setting "1.2," even though they are set in two different scenarios. For the **read_aocvm** command, AOCVM tables for library cell objects are applied in Scenario1, even though they are loaded in Scenario2.

```
prompt> set_app_var timing_library_derate_is_scenario_specific false

prompt> current_scenario Scenario1
prompt> set_timing_derate -max -late 1.1  [get_lib_cells lib_pvt5/AN2]

prompt> current_scenario Scenario2
prompt> set_timing_derate -max -late 1.2  [get_lib_cells lib_pvt5/AN2]
prompt> read_aocvm lib_cell_table.txt

prompt> current_scenario Scenario1
prompt> report_aocvm

****************************************
Report : aocvm
Design : top
Scenario(s): Scenario1
****************************************

                  |        | Fully      | Partially  | Not        |
                  | Total  | annotated  | annotated  | annotated  |
 -----------------+--------+------------+------------+------------+
  Leaf cells      |     6  |          6 |          0 |          0 |
  Nets            |     7  |          0 |          0 |          7 |
 -----------------+--------+------------+------------+------------+
                  |    13  |          6 |          0 |          7 |


1

prompt> report_timing_derate -scenario [list Scenario1 Scenario2]

****************************************
Report : timing derate
Design : top
Scenario(s): Scenario1 Scenario2
 ...
****************************************

    Design    Derate                                  value     Scenario
--------------------------------------------------------------------------------
```

```
    Cell      Derate                                  value     Scenario
-------------------------------------------------------------------------


    Lib Cell  Derate                                  value
-------------------------------------------------------------------------

    lib_pvt5/AN2
              clk_cell_delay_min_early                 -
              clk_cell_delay_min_late                  -
              clk_cell_delay_max_early                 -
              clk_cell_delay_max_late                1.20
              clk_cell_check_min_early                 -
              clk_cell_check_min_late                  -
              clk_cell_check_max_early                 -
              clk_cell_check_max_late                  -
              data_cell_delay_min_early                -
              data_cell_delay_min_late                 -
              data_cell_delay_max_early                -
              data_cell_delay_max_late               1.20
              data_cell_check_min_early                -
              data_cell_check_min_late                 -
              data_cell_check_max_early                -
              data_cell_check_max_late                 -
```

However, in the non-default state, with the variable set to true, each derate setting or the AOCVM table is specific to the scenario that is current when the constraint is applied:

```
prompt> set_app_var timing_library_derate_is_scenario_specific true
prompt> current_scenario Scenario1
prompt> set_timing_derate -max -late 1.1  [get_lib_cells lib_pvt5/AN2]

prompt> current_scenario Scenario2
prompt> set_timing_derate -max -late 1.2  [get_lib_cells lib_pvt5/AN2]
prompt> read_aocvm lib_cell_table.txt

prompt> current_scenario Scenario1
prompt> report_aocvm

****************************************
Report : aocvm
Design : top
Scenario(s): Scenario1
****************************************
No AOCVM derates.

1

prompt> report_timing_derate -scenario [list Scenario1 Scenario2]

****************************************
Report : timing derate
Design : top
Scenario(s): Scenario1 Scenario2
```

```
 ...
**************************************

    Design    Derate                                  value     Scenario
------------------------------------------------------------------------

    Cell      Derate                                  value     Scenario
------------------------------------------------------------------------

    Lib Cell  Derate                                  value     Scenario
------------------------------------------------------------------------

    lib_pvt5/AN2
              clk_cell_delay_min_early                 -        Scenario1
              clk_cell_delay_min_late                  -        Scenario1
              clk_cell_delay_max_early                 -        Scenario1
              clk_cell_delay_max_late                1.10       Scenario1
              clk_cell_check_min_early                 -        Scenario1
              clk_cell_check_min_late                  -        Scenario1
              clk_cell_check_max_early                 -        Scenario1
              clk_cell_check_max_late                  -        Scenario1
              data_cell_delay_min_early                -        Scenario1
              data_cell_delay_min_late                 -        Scenario1
              data_cell_delay_max_early                -        Scenario1
              data_cell_delay_max_late               1.10       Scenario1
              data_cell_check_min_early                -        Scenario1
              data_cell_check_min_late                 -        Scenario1
              data_cell_check_max_early                -        Scenario1
              data_cell_check_max_late                 -        Scenario1
    lib_pvt5/AN2
              clk_cell_delay_min_early                 -        Scenario2
              clk_cell_delay_min_late                  -        Scenario2
              clk_cell_delay_max_early                 -        Scenario2
              clk_cell_delay_max_late                1.20       Scenario2
              clk_cell_check_min_early                 -        Scenario2
              clk_cell_check_min_late                  -        Scenario2
              clk_cell_check_max_early                 -        Scenario2
              clk_cell_check_max_late                  -        Scenario2
              data_cell_delay_min_early                -        Scenario2
              data_cell_delay_min_late                 -        Scenario2
              data_cell_delay_max_early                -        Scenario2
              data_cell_delay_max_late               1.20       Scenario2
              data_cell_check_min_early                -        Scenario2
              data_cell_check_min_late                 -        Scenario2
              data_cell_check_max_early                -        Scenario2
              data_cell_check_max_late                 -        Scenario2
```

SEE ALSO

```
set_timing_derate(2)
read_aocvm(2)
report_timing_derate(2)
```

```
report_aocvm(2)
set_min_library(2)
```

timing_library_max_cap_from_lookup_table

Determines whether the tool should use the operating frequency to determine the maximum pin load.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable specifies whether the tool should use the operating frequency to determine the maximum pin load. The tool uses a single value for the maximum pin load by default.

When this variable is set to true, the tool uses the operating frequency to determine the maximum pin load.

To see the current value of this variable, use the **printvar timing_library_max_cap_from_lookup_table** command.

SEE ALSO

```
set_max_capacitance(2)
report_constraint(2)
```

timing_max_normalization_cycles

Sets an upper limit for the denominator when calculating normalized slack.

TYPE

integer

DEFAULT

4

DESCRIPTION

Normalized slack analysis is an optional analysis method that calculates normalized slack for each timing path:

```
normalized_slack = path_slack /
idealized_allowed_propagation_delay_for_path
```

The tool computes the allowed propagation delay for the path using ideal clock edges; it ignores setup time, uncertainty, and clock latency. The allowed propagation delay can be a half-cycle, a full cycle, or multiple cycles. It can be more complicated to compute when the launch and capture clocks are different.

The **timing_max_normalization_cycles** variable limits the runtime and memory used in this analysis by placing an upper limit on the denominator of the fraction used to calculate the normalized slack. The maximum allowed value of this denominator is this variable setting multiplied by the period of the launch clock for the path.

Setting a larger value supports a larger range of allowed propagation delays, possibly at the cost of more runtime and memory usage.

SEE ALSO

```
printvar(2)
report_timing(2)
timing_enable_normalized_slack(3)
```

timing_max_parallel_computations

Sets the maximum degree of parallelism used for parallel timing updates.

TYPE

integer

DEFAULT

0

GROUP

timing

DESCRIPTION

This application variable specifies the degree of parallelism used for parallel timing updates. Increasing the parallelism results in reduced runtime at the expense of memory.

When this variable has a value of **0** (the default), the number of cores used for timing update is specified by the value of the **-max_cores** option of the **set_host_options** command.

When this variable has a value of **1**, parallel timing update is disabled.

When this variable has a value that is larger than 1 but smaller than the value of the **set_host_options -max_cores** option, the number of cores being used must not exceed this limit and a smaller memory footprint is achieved.

When this variable has a value that is larger than the value of the **set_host_options -max_cores** option, the number of cores being used must not exceed the limit set with the **set_host_options** command and better runtime speedup can be achieved.

The actual degree of parallelism (number of processes/threads) must not exceed the value specified in this variable, but will not always be exactly the same. The tool derives the number internally to achieve the best performance.

SEE ALSO

```
set_host_options(2)
```

timing_pocvm_corner_sigma

Specifies the standard deviation used to calculate worst-case values from statistical parameters during parametric on-chip variation analysis.

TYPE

float

DEFAULT

3

GROUP

timing_variables

DESCRIPTION

Parametric on-chip variation (POCV) analysis internally computes arrival times, required times, and slack values based on statistical distributions. When performing comparisons between these statistical quantities, the tool needs to know what values in the distribution are considered worst-case.

The default behavior is to choose values at 3.0 standard deviations away from the mean value. In other words, for an arrival time distribution, the worst-case early arrival is 3.0 standard deviations below the mean, and the worst-case late arrival is 3.0 standard deviations above the mean.

You can use this variable set a different number of standard deviations away from the mean to determine the worst-case values for timing reports. Use a lower value such as 2.5 for less worst-case variation and more relaxed timing constraints. Conversely, use a higher value such as 3.5 for more worst-case variation and more restrictive timing constraints.

SEE ALSO

`timing_pocvm_enable_analysis(3)`

timing_pocvm_enable_analysis

Enables parametric on-chip variation (POCV) timing analysis.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

When this variable is set to **true**, the tool performs parametric on-chip variation (POCV) timing analysis. In this analysis mode, the tool internally computes arrival times, required times, and slack values as statistical distributions rather than fixed minimum and maximum values.

To determine the cumulative delay of a path, the POCV mode statistically combines the delay distribution of each stage. This is more accurate than simply adding the worst-case value from each stage. The resulting delay and slack values are more realistic and less pessimistic than values calculated by ordinary addition.

Before you can use POCV analysis, you must specify the statistical delay behavior of the logic gates used in the design and read that information into the tool using the **read_aocvm** command.

Note that POCV timing analysis uses more runtime than conventional timing analysis.

SEE ALSO

```
read_aocvm(2)
report_timing(2)
timing_pocvm_corner_sigma(3)
```

timing_remove_clock_reconvergence_pessimism

Enables or disables clock reconvergence pessimism removal.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable, when set to **true**, instructs the synthesis timing engine to remove 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 minimum and maximum delay of cells in the clock network.

Any effective change in the value of the **timing_remove_clock_reconvergence_pessimism** variable causes a full **update_timing**. You cannot perform one **report_timing** operation that considers CRP and one that does not without a full **update_timing** in between.

To run optimization with CRP removal, the clock network must be set as dont_touch:

```
prompt> set_app_var timing_remove_clock_reconvergence_pessimism true
true

prompt> report_timing
```

SEE ALSO

```
report_timing(2)
timing_crpr_threshold_ps(3)
timing_clock_reconvergence_pessimism(3)
```

timing_report_attributes

Specifies the list of attributes reported by the **report_timing -attributes** command.

TYPE

list

DEFAULT

{dont_touch dont_use map_only size_only ideal_net infeasible_paths}

GROUP

timing_variables

DESCRIPTION

This variable specifies the attributes reported by the **report_timing -attributes** command. The list can contain any of the following attributes: **dont_touch**, **dont_use**, **map_only**, **size_only**, **infeasible_paths** (Design Compiler and DC Explorer only), and **ideal_net**.

When you use the **-attributes** option of the **report_timing** command, the report shows the attributes that apply to each path increment under the heading "Attributes". A symbol key at the beginning of the path report lists the letter codes used for the attributes being reported. For example,

```
Attributes:
    d - dont_touch
    u - dont_use
   mo - map_only
   so - size_only
    i - ideal_net or ideal_network
  inf - infeasible path
```

To determine the current value of this variable, type **printvar timing_report_attributes**. For a list of all timing variables and their current values, type **print_variable_group timing**.

SEE ALSO

```
report_timing(2)
timing_variables(3)
```

timing_report_fast_mode

Enables the fast **report_timing** mode, which reports the worst timing paths among all path groups instead of individual path groups.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable, when set to true, selects the fast reporting mode for the **report_timing** and **get_timing_paths** commands.

When this variable is set to false (the default), the worst timing paths are reported separately for each path group. When this variable is set to true, the worst paths in the design are reported, irrespective of path groups, so fewer paths are reported.

Setting this variable to true results in the following changes in reporting behavior:

- The values specified with the **-max_paths** and **-nworst** options apply to all paths in the design rather than paths in each path group. For example, if the design has 12 path groups and **-max_paths** is set to 3, only 3 paths are reported instead of 36.
- The paths reported by the command are ordered strictly by slack and are not separated by path group.
- When the **-max_paths** and **-nworst** options are set to values larger than 1, only paths with negative slack are reported; paths with positive slack are omitted from the report.

Setting this variable to true results in the same reporting behavior as PrimeTime with respect to path groups and the **-max_paths** and **-nworst** options.

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

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

When this variable is set to **true** (the default), the tool counts a violating endpoint only once when computing the total negative slack (TNS) and the number of violating endpoints for a path group, scenario, or design. The TNS for a path group is the sum of worst violations for all endpoints in that path group. The TNS for a scenario is the sum of worst violations in that scenario for all endpoints, irrespective of the path group. The TNS for a design is the sum of worst violations for all endpoints, irrespective of path group or scenario.

When this variable is set to **false**, the tool computes the scenario TNS as the sum of TNS for all path groups in that scenario. The multi-scenario TNS is computed as the sum of TNS for all scenarios.

SEE ALSO

`report_qor(2)`

timing_save_library_derate

Controls whether the library cell derate settings are saved with the design (e.g., in milkyway cell or ddc files).

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls whether the library cell derate settings are saved with the design.

By default (false), these settings are not saved with the design, and need to be re-applied after re-opening the design in IC Compiler or Design Compiler. To save these settings with the design, set this variable to true before saving the design in IC Compiler or Design Compiler.

SEE ALSO

```
open_mw_cel(2)
save_mw_cel(2)
read_file(2)
write(2)
set_timing_derate(2)
```

timing_scgc_override_library_setup_hold

Specifies whether you can override library-specified clock-gating setup and hold values with new values specified with the **set_clock_gating_check** command.

TYPE

Boolean

DEFAULT

true

GROUP

timing

DESCRIPTION

When this variable is set to **true** (the default), you can use the **set_clock_gating_check** command to specify clock-gating setup and hold values on cells or pins, overriding the library-specified values on those cells or pins.

When this variable is set to **false**, the **set_clock_gating_check** command cannot override library-specified clock-gating setup and hold values on cells or pins.

Note that setup and hold values specified with the **set_clock_gating_check** command, when specified for clocks or the design, always have lower priority than library-specified values, irrespective of this variable setting. This variable only affects values set on cells or pins.

When the variable is set to **true** (the default), you can use the **set_clock_gating_check** command to set the setup and hold times to 0 for clock-gating checks at the design level, which enables automatic clock-gating check inferring and does not override any library values.

SEE ALSO

```
set_clock_gating_check(2)
report_clock_gating(2)
```

```
report_clock_gating_check(2)
report_timing(2)
```

timing_self_loops_no_skew

Affects the behavior, runtime, and CPU usage of **report_timing** and **compile**.

Note: This variable will be obsolete in the next release. Please adjust your scripts accordingly.

TYPE

Boolean

DEFAULT

false

GROUP

timing_variables

DESCRIPTION

Affects the behavior, runtime, and CPU usage of **report_timing** and **compile**. When set to *true*, clock skew is eliminated for a path that starts and ends at the same register. When set to *false* (the default value), clock skew is not eliminated. Thus, the timing for such paths is pessimistic. To obtain the more accurate behavior of no clock skew (uncertainty) for such paths, set this variable to *true*. However, note that runtime and memory usage might increase significantly.

To determine the current value of this variable, type **printvar timing_self_loops_no_skew**. For a list of all **timing** variables and their current values, type **print_variable_group timing**.

SEE ALSO

```
report_timing(2)
timing_variables(3)
```

timing_separate_clock_gating_group

Specifies if a separate cost group is used for clock-gating checks in timing analysis, reports, and optimization.

TYPE

Boolean

DEFAULT

false

GROUP

timing

DESCRIPTION

When this variable is set to **true**, a separate cost group named ***clock_gating_default*** is created for all clock-gating checks.

When set to **false** (the default), the clock-gating check is applied to the cost group of the clock being gated.

If multiple scenarios exist, a cost group is created for clock-gating checks for each scenario when this variable is **true**.

You can change the weight and critical range settings for this cost group using the **group_path** command with the **-name** ***clock_gating_default*** option.

SEE ALSO

```
get_path_groups(2)
group_path(2)
report_clock_gating(2)
report_clock_gating_check(2)
report_constraint(2)
report_path_group(2)
report_qor(2)
report_timing(2)
```

timing_through_path_max_segments

Controls the maximum number of latches for reporting paths through latches.

TYPE

Integer

DEFAULT

5

DESCRIPTION

When **timing_through_path_max_segments** is set to a non-zero 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**, limiting the reporting on the long path but speeding up analysis. With a small non-zero value, many transparent latch data pins in the design will be selected. With a larger value, fewer will be selected.

When **timing_through_path_max_segments** is set to 0, no latch data pins are selected. Reporting on paths through many latches is allowed, but analysis may be slower.

The variable **timing_through_path_max_segments** has no effect if the variable **timing_enable_through_paths** is set to **false**.

To determine the current setting, use the following command:

```
printvar timing_through_path_max_segments
```

SEE ALSO

```
printvar(2)
report_timing(2)
timing_enable_through_paths(3)
```

timing_use_ceff_for_drc

Specifies whether to use effective capacitance (Ceff) or total capacitance (Ctot) for DRC fixing of maximum capacitance violations.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable set to **false** (the default), total capacitance (Ctot) values are compared against the maximum capacitance limit during DRC fixing. When this variable is set to **true**, effective capacitance (Ceff) values are used instead.

Total capacitance is the sum of all the capacitance values in an RC network. Effective capacitance is a lumped value that results in a similar delay effect as the full RC network, calculated as the average capacitive loading that a gate observes at the delay trip-point.

The effective capacitance can be very different from the maximum total capacitive loading experienced by the gate. Choosing to use effective capacitance can possibly result in extrapolation errors.

Setting the variable to **true** may result in less pessimistic DRC fixing. Before you set this variable to **true**, be sure to carefully evaluate your capacitance fixing and signoff methodology.

SEE ALSO

```
report_delay_calculation(2)
set_max_capacitance(2)
```

timing_use_clock_specific_transition

Propagate the transition from the specific clock path for latency calculation.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** (the default value), the tool only propagates the transition of the clock path, for the purpose of clock latency calculation. If there are multi-input gates on the clock network, the transition of non-clock inputs are ignored during clock latency calculation.

For generated clocks defined on an output of a gate, the tool propagates the transition from the path connected to its master clock, and uses that transition value for the purpose of calculating the clock latency for the generated clock.

When set to **false**, the tool allow transition from the non-clock input of multi-input gates along the clock path to be used for clock latency calculation.

In addition, when set to **false**, the generated clock defined at the output of a gate uses zero transition at the generated clock source.

timing_use_driver_arc_transition_at_clock_source

Uses the backward cell arc to compute a realistic driver model at the driver pin for primary clock sources and also a generated clock that cannot trace back to its master clock.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When set to **true** (the default value), the tool uses the backward cell arcs, (when at least one exists), to compute a worst-case driver model. This behavior applies to the primary clock sources, which are defined by the **create_clock** command, and generated clock sources (defined by the **create_generated_clock** command) that cannot trace back to its master clock.

When set to **false** the tool asserts a zero-transition, ideal ramp model at the driver pin.

SEE ALSO

```
create_clock(2)
create_generated_clock(2)
```

timing_use_enhanced_capacitance_modeling

Specifies whether to use the enhanced capacitance modeling information available in the library description of a pin.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to **true** (the default), the tool uses the enhanced capacitance modeling information available in the library description of a pin, consisting of different capacitance values for rise and fall transitions, and possibly a range of worst-case values for each type of transition. This setting is recommended for better timing correlation with the PrimeTime tool.

When this variable is set to **false**, the tool uses the single default capacitance value specified in the library description of the pin and ignores any enhanced capacitance information.

The library description of pin in Liberty format can contain both a default capacitance value and more specific values or ranges of capacitance values for rise and fall transitions. For example,

```
pin("IN1") {
  direction : input;
  capacitance : 0.0067542;
  rise_capacitance : 0.0123321;
  fall_capacitance : 0.0056745;
  rise_capacitance_range (0.0045670, 0.012345);
  fall_capacitance_range (0.005656, 0.0123123);
}
```

When the variable is set to **true**, the tool uses the worst-case capacitance values specified by the rise and fall range attributes (for example, the high value for late arrival timing and the low value for early arrival timing). When the variable is set to **false**, the tool uses the single value specified by the plain "capacitance" attribute and ignores the rise and fall capacitance attributes.

SEE ALSO

`set_operating_conditions(2)`

timing_variables

Variables that affect timing.

SYNTAX

```
Boolean disable_library_transition_degradation = false
Boolean disable_auto_time_borrow = false
Boolean timing_self_loops_no_skew = false
list timing_report_attributes = {"dont_touch" "dont_use" "map_only"
"size_only"}
Boolean disable_case_analysis = false
Boolean case_analysis_with_logic_constants = false
String case_analysis_log_file = ""
Boolean enable_slew_degradation = "false"
Boolean high_fanout_net_threshold = "1000"
```

DESCRIPTION

These variables directly affect timing aspects in **dc_shell**. Defaults are shown under Syntax.

For a list of timing_variables, type **print_variable_group timing**. To view this manual page online, type **help timing_variables**. To view an individual variable description, type **help *var*, where *var* is the name of the variable.**

disable_auto_time_borrow

Affects automatic time borrowing. When **disable_auto_time_borrow** is **false** (the default value), automatic time borrowing balances the slack along back-to-back latch paths to reduce the overall delay cost. When **true**, no slack balancing will occur during time borrowing.

disable_library_transition_degradation

When **false** (the default), **report_timing** and other commands that use timing in dc_shell use the transition degradation table in the library to determine the net transition time. When **true**, the timing commands behave as though there were no transition degradation across the net.

disable_library_transition_degradation

When **false** (the default), **report_timing** and other commands that use timing in dc_shell try to balance the slack between paths. When **true**, the cycle time is portioned such that the first path attempts to make timing and the second path only gets the remainder of the delay.

timing_self_loops_no_skew

This Boolean variable affects the behavior, runtime, and cpu usage of **report_timing** and **compile**. When **true**, register. When **false** (the default value), clock skew is not eliminated; thus, the timing for such paths is pessimistic. To obtain the more accurate behavior of no clock skew (uncertainty) for such paths, set the variable to **true**, but note that runtime and memory usage may increase significantly.

timing_report_attributes

Specifies the list of attributes to be reported with **report_timing -attributes**. The current list of attributes supported are dont_touch, dont_use, map_only, and size_only.

disable_case_analysis

Determines whether constant propagation is performed in the design from pins either that are tied to a logic constant value, or for which a **case_analysis** command is specified. By default, constant propagation happens if the **set_case_analysis** is specified or if the variable case_analysis_with_logic_constants is set to true, unless the variable disable_case_analysis is set to false.

case_analysis_with_logic_constants

Determines whether constant propagation will be preformed if there are only logic constants in the circuit and there is no set_case_analysis command used. By default, logic constants are not propagated in the absence of set_case_analysis commands or case_analysis_with_logic_constants being set to true.

case_analysis_log_file

Specifies the name of a log file to be generated during propagation of constant values from case analysis or from nets tied to logic zero or to logic one. The log file contains the list of all ports and pins that propagate constants.

enable_slew_degradation

When **false** (the default setting), optimization with physical information does not consider transition degradation across a net and only tries to fix transition violations on a per net basis. Setting this variable to **true** will cause optimization with physical information to consider transition degradation across a net and try to fix transition violations on a per cell basis.

high_fanout_net_threshold

Specifies the minimum number of loads for a net to be classified as a high-fanout net, the default is 1000. The delays and loads of high-fanout are computed using a simplified model assuming a fixed fanout number. The rationale behind this is that delays of high-fanout nets are expensive to compute but such nets are often unconstrained (as in the case of global reset nets, scan enable nets, and so on). So detailed delay calculations on such nets are expensive and usually unnecessary.

SEE ALSO

```
create_clock(2)
report_timing(2)
disable_library_transition_degradation(3)
timing_self_loops_no_skew(3)
timing_report_attributes(3)
disable_case_analysis(3)
case_analysis_with_logic_constants(3)
case_analysis_log_file(3)
enable_slew_degradation(3)
high_fanout_net_threshold(3)
```

timing_waveform_analysis_mode

Controls usage of CCS-based waveform analysis for timing calculation.

TYPE

string

DEFAULT

disabled

GROUP

timing_variables

DESCRIPTION

Advanced geometry nodes at 28 nm and below can be significantly impacted by waveform distortion effects. The tool offers an advanced, gate-level waveform propagation analysis mode that captures the effects of such distortion on the next stage delay, including the Miller effect and long tail effect, by propagating the transition waveform along with the delay and transition time.

The waveform propagation mode uses the CCS timing models in the logic libraries and uses propagated piecewise linear waveforms in place of simplified equivalent waveforms. It produces more accurate results when the voltage waveform shapes differ significantly from the library characterization driver waveform shapes. The CCS modeling data must be available in the logic libraries to get the more accurate results.

You control this feature with the **timing_waveform_analysis_mode** variable. The default setting is **disabled**. Set this variable to **clock_network** to apply CCS-based waveform propagation analysis to the clock network only, or **full_design** to apply it to the whole design. Runtime and memory usage might increase with this feature enabled.

By default, the waveform propagation mode uses CCS timing models, but not CCS noise models. You can optionally add usage of CCS noise models for even higher accuracy. To do so, set the **timing_enable_ccsn_waveform_analysis** variable to **true**.

In the PrimeTime tool, you can control the same CCS-based waveform analysis feature by setting the **delay_calc_waveform_analysis_mode** variable to **clock_network** or

full_design. However, the PrimeTime tool does not have a separate variable to control usage of CCS noise models. PrimeTime version I-2013.12 (or later) uses CCS noise models, whereas PrimeTime version H-2013.06 does not. For good timing correlation between the two tools, set the variables to select the same waveform analysis options.

SEE ALSO

```
report_timing(2)
update_timing(2)
```

tio_allow_mim_optimization

Enables support for multiply instantiated modules (MIMs) during interface optimization performed using the **route_opt** and **focal_opt** commands.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable enables support for multiply instantiated modules (MIMs) during interface optimization performed using the **route_opt** and **focal_opt** commands.

When interface optimization is enabled on MIMs,

1. The tool optimizes inside MIMs.
2. The tool ensures that optimization done in any instance improves timing across all the instances of the MIM.
3. The tool replicates the netlist changes across all the instances.
4. At the end of optimization, the reference cell is taken through the block update steps consisting of the **legalize_placement**, **route_zrt_eco**, and **create_block_abstraction** commands.

When this variable is set to **false**, the tool cannot optimize the interface of blocks that are multiply instantiated.

SEE ALSO

`focal_opt(2)`

```
route_opt(2)
set_top_implementation_options(2)
```

tio_eco_output_directory

This variable specifies the name of directory used for ECO file generated during interface optimization feature in IC Compiler.

TYPE

string

DEFAULT

TIO_eco_changes

DESCRIPTION

tio_eco_output_directory provides option for users to change the directory name for ECO file generated during interface optimization feature in route_opt. This variable is applicable only when ECO file generation feature is enabled by setting TcL variable **tio_write_eco_changes** to true.

When **tio_eco_output_directory** is specified tool writes out, into the directory specified, an ECO file for each block with file name <block_reference_name>.tcl. If the specified directory name already exists on disk or have files with <block_reference_name>.tcl, tool reuses the directory and the files (if any) and append to the existing files. This is useful when running multiple rounds of interface optimization and users wants to have the ECO changes consolidated into one file per block. User can change the **tio_eco_output_directory** before running each round of route_opt with interface optimization to have ECO files into the required directories.

If user has not specified the directory name, tool uses **TIO_eco_changes** as the directory name. If this directory exists tool adds suffix .1, .2 etc. so as to make it unique.

For example, user runs the following steps without setting this variable route_opt [with TIO] -- first route_opt route_opt inc [with TIO] -- second route_opt

ECO changes are written into files under TIO_eco_changes below for the first route_opt TIO_eco_changes/<block1_reference_name>.tcl TIO_eco_changes/ <block2_reference_name>.tcl And files under TIO_eco_changes.1 below for second route_opt TIO_eco_changes.1/<block1_reference_name>.tcl TIO_eco_changes.1/ <block2_reference_name>.tcl

If user specifies directory name with this variable like below. set tio_eco_output_directory TIO_eco_changes_pass1 route_opt [with TIO] -- first route_opt route_opt inc [with TIO] -- second route_opt

ECO changes are written into files under TIO_eco_changes_pass1 for the first route_opt as below TIO_eco_changes_pass1/<block1_reference_name>.tcl TIO_eco_changes_pass1/<block2_reference_name>.tcl And ECO changes are appended to the files generated in first route_opt under TIO_eco_changes_pass1 without new directory creation. TIO_eco_changes_pass1/<block1_reference_name>.tcl -- append TIO_eco_changes_pass1/<block2_reference_name>.tcl -- append

If the user specified a directory name with this variable, the user needs to set this variable to be different directory to prevent the tool from appending the changes to the same directory. set tio_eco_output_directory TIO_eco_changes_pass1 route_opt [with TIO] -- first route_opt set tio_eco_output_directory TIO_eco_changes_pass2 route_opt inc [with TIO] -- second route_opt

ECO changed are written into files under TIO_eco_changes_pass1 for the first route_opt TIO_eco_changes_pass1/<block1_reference_name>.tcl TIO_eco_changes_pass1/<block2_reference_name>.tcl And under TIO_eco_changes_pass2 for the second route_opt TIO_eco_changes_pass2/<block1_reference_name>.tcl TIO_eco_changes_pass2/<block2_reference_name>.tcl

If the directory/ECO file specified already existed and user doesn't have write permission, it gives an error message and does not write the ECO file. User need to check and change the permission or set to another name with this variable.

SEE ALSO

```
tio_write_eco_changes(3)
tio_skip_block_update(3)
route_opt(2)
set_top_implementation_options(2)
```

tio_preload_block_site_rows

Specifies the IC Compiler to load sub-block site rows during initial linking in flows involving transparent interface optimization (TIO). This variable is to be used for designs with different site array configurations in top and sub-blocks.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Setting this variable as **true** makes IC Compiler to load the sub-block site rows during initial linking of top-level design. This is useful for flows that need to process site rows and site types of both Top-level and sub-blocks. An example usage is when running transparent interface optimization (TIO) on designs having different site rows in top and sub-blocks.

To use this feature, you must set the variable to **true** in the IC Compiler tool before you open the design.

SEE ALSO

```
set_top_implementation_options(2)
check_legality(2)
```

tio_preserve_routes_for_block

Setting this variable to true enables zroute based route preservation inside the blocks when transparent interface optimization is performed using the route_opt or focal_opt commands.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

During transparent interface optimization, when the tool changes a cell inside a block abstraction, the net identities can also change. For example, when inserting a buffer creates a new net, although the tool seeks to place the new cells along the route and maintain route topology, it does not simultaneously reassign the net identity of the route metallization geometries associated with the net. Route preservation performs this task in a process executed after the optimization phase finishes and before ECO routing begins. The presence of the route segments bound to the nets assist the ECO router to find a similar solution to the one obtained before optimization. In this way, preserving the routes improves correlation of the routing before and after optimization and therefore acts to improve the convergence in timing before and after ECO.

This variable is only active when Zroute is selected. If Zroute is disabled by using the set_route_mode_options -zroute false command, this variable is not active.

SEE ALSO

`route_opt(2)`

tio_skip_block_update

Skip block update of blocks, specified through set_top_implementation_options, during transparent interface optimization performed using the route_opt and focal_opt commands.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

By default, all the blocks opened for transparent interface optimization, by using the set_top_implementation_options command, are taken through a block update step consisting of placement legalization, eco routing and block abstraction steps.

Setting this variable to true makes IC Compiler skip the block update step. This is useful in case you want to skip the block update and instead write out an ECO file that can be used later to update the block.

In route_opt, block update is skipped only if the **route_opt** command is run in incremental mode with low or medium effort.

SEE ALSO

```
set_top_implementation_options(2)
route_opt(2)
focal_opt(2)
```

tio_write_eco_changes

Controls how the optimization changes to blocks are written into ECO files when transparent interface optimization is performed.

TYPE

Boolean

DEFAULT

false

GROUP

routeopt_variables

DESCRIPTION

This variable controls how the optimization changes to blocks are written into ECO files when interface optimization is performed. This is useful for cases where you cannot wait until all blocks complete transparent interface optimization due to varying block sizes and complexity, or when block owners have a requirement to merge with other changes that might include design and timing ECOs to save overall turnaround time. Use this variable with the **tio_skip_block_update** variable to skip the block update step of transparent interface optimization.

When the **tio_write_eco_changes** variable is set to **true**, the optimization changes are written into a file in the directory named TIO_eco_changes with <block_reference_name>.tcl as the file name. If a directory named TIO_eco_changes already exists, the tool adds a suffix of .1, .2, .3, etc., to make the directory name unique. The ECO output directory name can be customized by using the **tio_eco_output_directory** variable.

The runtime and memory usage might increase slightly when this variable is set to **true**, due to the extra file generation.

SEE ALSO

```
route_opt(2)
set_top_implementation_options(2)
tio_eco_output_directory(3)
tio_skip_block_update(3)
```

trackAssign_XTalkParam

DESCRIPTION

The **trackAssign_XTalkParam** variable defines the cost for crosstalk prevention. Use this variable when is enabled.

SYNTAX

```
      set trackAssign_XTalkParam value
```

```
The valid values of this variable range between 1 and 4.
The default is 1.
```

EXAMPLE

To define the cost for crosstalk prevention, enter

set trackAssign_XTalkParam 3

trackAssign_densityDriven

DESCRIPTION

The **trackAssign_densityDriven** variable specifies the mode for density-driven track assignment.

When set to -1 (the default), the router tries to spread the wires looking at one extra track if timing or crosstalk is enable.

When set to 0, the router does not perform spreading.

When set to 1, the router tries to spread wires looking at one extra track.

When set to 2, the router tries to spread wires looking at two extra tracks. You should set this mode only when the utilization of the design is less than 60 percent because this mode uses more runtime. Using this mode when the design is not sparse can have a negative impact.

SYNTAX

```
set trackAssign_densityDriven value
```

EXAMPLE

To spread wires looking at one extra track, enter

set trackAssign_densityDriven 1

trackAssign_evenSpaceAdjustment

DESCRIPTION

The **trackAssign_evenSpaceAdjustment** variable specifies the mode of cost calculation for wire spreading.

When set to 1, the router tries every track sequentially.

When set to 2 (the default), the router tries every other track.

When set to 3, the router tries every third track first.

SYNTAX

```
set trackAssign_evenSpaceAdjustment value
```

EXAMPLE

To try every third track first, enter

set trackAssign_evenSpaceAdjustment 3

trackAssign_minimizeJog

DESCRIPTION

The **trackAssign_minimizeJog** variable specifies the mode to minimize jogs.

When set to 0 (the default), the router performs normal operation.

When set to 1, the router minimizes jobs near non-standard-cell pins.

SYNTAX

```
set trackAssign_minimizeJog value
```

EXAMPLE

To minimize jogs near the nonstandard-cell pins, enter

set trackAssign_minimizeJog 1

trackAssign_noOffGridRouting

DESCRIPTION

The **trackAssign_noOffGridRouting** variable specifies the switch for off-grid routing on macro pins.

When set to 0 (the default), the router automatically uses off-grid routing for macro pins. When set to 1, the router puts all wires on-grid.

SYNTAX

```
set trackAssign_noOffGridRouting value
```

EXAMPLE

To put all the wires on grid, enter

set trackAssign_noOffGridRouting 1

trackAssign_noiseThreshold

DESCRIPTION

The **trackAssign_noiseThreshold** variable specifies the noise threshold for track assignment.

SYNTAX

```
     set trackAssign_noiseThreshold value
```

```
The valid values of this variable range between 0.000 and
1.000. The default is 0.450.
```

EXAMPLE

To specify the noise threshold for track assignment as 0.350, enter

set trackAssign_noiseThreshold 0.350

trackAssign_parallelLimit

DESCRIPTION

The **trackAssign_parallelLimit** variable is used to specify parallel lengths limit. Use this variable when is enabled.

SYNTAX

```
set trackAssign_parallelLimit value
```

The valid values of this variable range between 0 and 10000000. The default is 0.

EXAMPLE

To specify parallel length limit as 500, enter

set trackAssign_parallelLimit 500

trackAssign_parallelLimitMode

DESCRIPTION

The **trackAssign_parallelLimitMode** variable is set to break long wire mode on or off.

When set to 0 (the default), track assignment does not check parallel lengths. When set to 1, track assignment breaks a long wire if it exceeds the parallel length or the capacitance limit.

SYNTAX

```
set trackAssign_parallelLimitMode value
```

EXAMPLE

To ignore breaking long wires even when parallel length or capacitance limit is exceeded, enter

set trackAssign_parallelLimitMode 0

trackAssign_runTimingMode

DESCRIPTION

The **trackAssign_runTimingMode** variable enables or disables timing-driven track assignment.

When set to 0 (the default), timing-driven track assignment is disabled. When set to 1, timing-driven track assignment is enabled.

SYNTAX

```
set trackAssign_runTimingMode value
```

EXAMPLE

To perform timing-driven track assignment, enter

set trackAssign_runTimingMode 1

trackAssign_runXTalkIter

DESCRIPTION

The **trackAssign_runXTalkIter** variable allows you to specify the number of extra iterations for crosstalk during track assignment. The default number of iterations are determined automatically. Use this variable when is enabled.

SYNTAX

```
set trackAssign_runXTalkIter value
```

```
The valid values of this variable range between -1 and 5.
The default is -1.
```

EXAMPLE

To specify the number of extra iterations for crosstalk in track assignment, enter axSetIntParam "trackAssign" "trackAssign_runXTalkIter" 4

trackAssign_runXTalkMode

DESCRIPTION

The **trackAssign_runXTalkMode** variable is used to enable or disable the crosstalk mode during track assignment.

When set to 0 (the default), crosstalk mode is disabled. When set to 1, crosstalk mode is enabled and track assignment minimizes crosstalk induced delay and noise.

SYNTAX

```
set trackAssign_runXTalkMode value
```

EXAMPLE

To minimize crosstalk-induced delay and noise, enter

set trackAssign_runXTalkMode 1

trackAssign_timingCost

DESCRIPTION

The **trackAssign_timingCost** variable sets the weight given to timing relative to the wire length during track assignment. This variable is effective only when is set to 1, which runs track assignment in timing-driven mode.

SYNTAX

```
set trackAssign_timingCost value
```

```
The valid values of this variable range between 1 and 10.
The default is 1.
```

EXAMPLE

To specify the cost for timing optimization, enter

set trackAssign_timingCost 6

trackAssign_tryGlobalLayerFirst

DESCRIPTION

The **trackAssign_tryGlobalLayerFirst** variable specifies the mode to use the global layer first.

When set to 0, track assignment does not try to use the global layer. When set to 1 (the default), track assignment tries to use the global layer in the first iteration.

SYNTAX

```
set trackAssign_tryGlobalLayerFirst value
```

EXAMPLE

To ignore using global layer, enter

set trackAssign_tryGlobalLayerFirst 0

trackAssign_tryGlobalLayerOnly

DESCRIPTION

The **trackAssign_tryGlobalLayerOnly** variable specifies the mode to use only global layer during track assignment.

When set to -1 (the default), track assignment tries the global layers only if timing-driven or crosstalk-driven track assignment is enabled.

When set to 0, track assignment can try layers other than global layers in the second phase.

When set to 1, track assignment tries only the global layers.

SYNTAX

```
set trackAssign_tryGlobalLayerOnly value
```

trackAssign_variableWidthAdjustment

DESCRIPTION

The **trackAssign_variableWidthAdjustment** variable specifies the width adjustment for fat wire tracks. The calculation for the number of tracks that a fat wire occupies is sometime too conservative; that is, track assignment reserves too many tracks for each fat wire. The **trackAssign_variableWidthAdjustment** variable can help reduce the demand and pack the fat wires more tightly.

You can specify an integer between 0 and 5, which specifies how many tracks to reduce the detail routing track count for track assignment. The default is 0 (track assignment uses the detail routing track count).

SYNTAX

```
set trackAssign_variableWidthAdjustment value
```

EXAMPLE

To specify the width adjustment for fat wire tracks, enter

set trackAssign_variableWidthAdjustment 2

track_attributes

Contains attributes related to track.

DESCRIPTION

Contains attributes related to track.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified track. Specified with **list_attribute -class track -application**, the definition of attributes can be listed.

Track Attributes

bbox

Specifies the bounding-box of a track. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The *bbox* of a track is calculated by the **origin** and **orientation** of its cell and the actual **bbox** of its corresponding terminal from the child MW design.

This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding-box of a track.

The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ll** of a track by accessing the first element of its **bbox**.

This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a track.

The data type of **bbox_llx** is double.

This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a track.

The data type of **bbox_lly** is double.

This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a track.

The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ur** of a track by accessing the second element of its **bbox**.

This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a track.

The data type of **bbox_urx** is double.

This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a track.

The data type of **bbox_ury** is double.

This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a track object is located.

This attribute is read-only.

count

Specifies the number of grid of a track.

The data type of **count** is integer.

This attribute is read-only.

direction

Specifies the routing direction of a track.

The valid values can be: X, Y.

This attribute is read-only.

layer

Specifies the layer name where a track object is located on.

This attribute is read-only.

name

Specifies the object name of a track.

This attribute is read-only.

object_class

Specifies object class name of a track, which is **track**.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

This attribute is read-only.

reserved_for_width

Specifies if a track is reserved for specified width.

The data type of **reserved_for_width** is boolean.

This attribute is read-only.

space

Specifies track step of a track.

The data type of **space** is integer.

This attribute is read-only.

start

Specifies start position of a track.

The data type of **start** is coordinate point.

This attribute is read-only.

stop

Specifies end position of a track.

The data type of **stop** is coordinate point.

This attribute is read-only.

width

Specifies the width of wires associated with the track.

The data type of **width** is double.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
```

transfer_reserved_metalblockage_within_pin_to_viablockage

Transfers the system metal blockage layer within pins to the via blockage layer during FRAM view creation.

TYPE

Boolean

DEFAULT

false

GROUP

extract_blockage_pin_via_variables

DESCRIPTION

This variable, when set to true, transfers the system metal blockage layer within pins to the via blockage layer during execution of the **extract_blockage_pin_via** command. This option works only when the **-preserve_all_metal_blockage** option is used in the **extract_blockage_pin_via** command.

SEE ALSO

`extract_blockage_pin_via(2)`

ungroup_keep_original_design

Controls whether the **ungroup** command keeps the original design when a design is ungrouped.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

By default, the **ungroup** command deletes the original design if all the instances of a design have been ungrouped and there are no other references to the design.

When set to **true**, the original design is preserved.

For example, assume there are two instances of the *mid* design named *mid1* and *mid2*. If you run the **ungroup -flatten -all** command, after the tool collapses the hierarchies, the design called *mid* is deleted from memory if there are no other references to the design. This variable is used to change the behavior. When this variable is set to **true**, the ungrouped design is preserved.

SEE ALSO

```
set_ungroup(2)
ungroup(2)
```

uniquify_keep_original_design

Controls the **uniquify** command to keep the original design when a multiply-instantiated design is uniquified.

TYPE

Boolean

DEFAULT

false

GROUP

none

DESCRIPTION

By default, the **uniquify** command deletes the original design if all the instances of a design have been uniquified and there are no other references to the design from anywhere else.

When set to **true**, the original design is preserved.

For example, if there are two instances of the *mid* design, uniquify creates two new designs named *mid_1* and *mid_2*. By default the original design named *mid* is deleted. This variable is used to change this behavior.

SEE ALSO

`uniquify(2)`

uniquify_naming_style

Specifies the naming convention to be used by the **uniquify** command.

TYPE

string

DEFAULT

%s_%d

GROUP

system_variables

DESCRIPTION

This variable specifies the naming convention to be used by the **uniquify** command. The variable string must contain only one %s (percent s) and one %d (percent d) character sequence. To use a percent sign in the design name, two are needed in the string (%%).

SEE ALSO

```
uniquify(2)
system_variables(3)
```

update_clock_latency_consider_float_pin_delays

Controls whether the **update_clock_latency** command should consider float pin delay values at clock sinks or not.

TYPE

Boolean

DEFAULT

false

GROUP

update_clock_latency_variables

DESCRIPTION

The **update_clock_latency** command determines typical latencies for the real clocks in a design and transfers these latencies to other clocks as per **set_latency_adjustment_options** settings. Use the **update_clock_latency_consider_float_pin_delays** variable to indicate whether float pin delay values should be considered in calculating the typical latency or not. The float pin delay values can originate from manual settings using the **set_clock_tree_exceptions -float_pins** command, or can be generated using the **skew_opt** command in a concurrent clock and data (CCD) flow.

By default, float pin delay values are ignored when calculating the typical latency of the real clocks. The considered latency at any sink is then equal to the latency from the clock source to the sink. However, when the **update_clock_latency_consider_float_pin_delays** variable is set to 'true', the considered latency at a clock sink is calculated as the latency from the clock source to the sink *plus* the float pin delay value. To arrive at the typical latency value of a clock, an averaging strategy of the considered latency with outlier filtering over all sinks of the clock is applied.

To determine the current value of the **update_clock_latency** variables, type **printvar update_clock_latency***.

SEE ALSO

```
set_latency_adjustment_options(2)
set_clock_tree_exceptions(2)
skew_opt(2)
update_clock_latency(2)
```

upf_allow_DD_primary_with_supply_sets

Allows using domain dependent supply nets of the design as the primary supply of the power domain when supply sets are used in the design.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When supply sets are used in a design, domain dependent supply nets cannot be used as the primary supply of the power domain. Setting this variable to **true** removes this restriction. The default value of this variable is **false**.

After you read the UPF file, the tool ignores any changes to this variable setting. To change the variable setting, remove the UPF file, change the value of the variable, and reload the UPF file.

SEE ALSO

`load_upf(2)`

upf_charz_allow_port_punch

Allows UPF-related port punching to occur within the **characterize** command on the blocks being characterized.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

The **upf_charz_allow_port_punch** variable, when set to **true**, allows the **characterize** command to modify the block interfaces to automatically port punch the control signals of the power management cells from the top design and bring them into the block.

SEE ALSO

`characterize(2)`

upf_charz_enable_supply_port_punching

Allows UPF-related supply port punching to occur within the **characterize** command on the blocks being characterized.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

The **upf_charz_enable_supply_port_punching** variable, when set to **true**, allows the **characterize** command to modify the block interfaces to automatically punch supply ports to bring required supply nets into the block if they are not already available.

SEE ALSO

`characterize(2)`

upf_charz_max_srsn_messages

Sets the maximum number of error and warning messages related to the **set_related_supply_net** command that can be printed when characterizing a block.

TYPE

int

DEFAULT

10

GROUP

upf

DESCRIPTION

To reduce the verbosity of the **characterize** command, a limit is set on the number of messages related to the **set_related_supply_net** command that can be printed during characterization. You can change the verbosity of these messages by assigning an integer value to this variable.

SEE ALSO

```
characterize(2)
set_related_supply_net(2)
```

upf_create_implicit_supply_sets

Allows creation of supply set handles for the power domains.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable is set to **true** to enable the creation of supply set handles. When this variable is **true**, the tool creates the primary, default_isolation, and default_retention supply set handles when creating the power domains. Other user-defined supply set handles defined with the **-supply** option of the **create_power_domain** command are also created.

Note:
You must set this variable before creating the power domains. After creating the power domains, this variable is considered read-only. The tool issues the **CMD-013** error message if you change the value of the variable after creating the power domain.

You should set the variable at the beginning of the script, before reading in the design with commands such as **open_mw_cel**, **import_designs**, **read_verilog**, and **read_ddc**. Do not change the variable setting for the rest of the IC Compiler session.

SEE ALSO

`create_power_domain(2)`

upf_enable_legacy_block

Allows UPF 1.0-style design blocks to be integrated into a UPF 2.0-style design.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

The **upf_enable_legacy_block** variable, when set to **true**, allows the **legacy_block** attribute to be applied to a UPF 1.0-style design instance through the **set_design_attribute** command. The **legacy_block** attribute on a UPF 1.0-style design instance allows a UPF 2.0-style domain-independent supply net outside the block to be connected to a domain-dependent net inside at the block boundary. The connection of a domain-independent supply net to a domain-dependent one is illegal otherwise.

SEE ALSO

upf_enable_relaxed_charz

Allows flexible partitioning of power domain and supports **-location parent** for isolation cells in a hierarchical flow.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

The **upf_enable_relaxed_charz** variable, when set to **true**, allows the **allocate_fp_budgets** command to handle **-location parent** for isolation cells and also allows greater flexibility in partitioning the design. **Budgeting** can partition a part of power domain.

SEE ALSO

`allocate_fp_budgets(2)`

upf_extension

Disables writing UPF extension commands in the **save_upf** command.

TYPE

Boolean

DEFAULT

true

GROUP

mv

DESCRIPTION

This variable controls whether the **save_upf** command writes UPF extension commands such as **set_related_supply_net** into UPF files. By default, UPF extension commands are written by **save_upf** in the following format:

```
if {[info exists upf_extension] && upf_extension} {
<upf_extension_command>
}
```

If **upf_extension** is set to **false**, these lines are not written by **save_upf**. The UPF extension command itself continues to be written by the **write_script** command, and continues to be readable by the **source** and **load_upf** commands.

All tools, including third party tools, that need to support the UPF extension commands must predefine the **upf_extension** Tcl variable as **true**.

SEE ALSO

`set_related_supply_net(2)`

upf_levshi_on_constraint_only

Inserts level shifters only at the domain boundaries with a level shifter strategy.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls whether level shifter cells can be inserted at power domain boundaries without a strategy. By default, level shifters can be inserted at domain boundaries without a strategy. This gives more flexibility to the tool and has better QoR.

If you want to restrict the tool to insert level shifters only at the boundaries where the **set_level_shifter** command is specified, set the variable to **true**. The tool will not insert level shifters at all of the boundaries with **set_level_shifter**. The tool can insert level shifters only at the boundaries with the **set_level_shifter** constraint.

SEE ALSO

`set_level_shifter(2)`

upf_name_map

Specifies the name map files to be used during reapplication of golden UPF to designs.

TYPE

list

DEFAULT

""

GROUP

mv

DESCRIPTION

This variable specifies the name of the mapping file to be used for each design during reapplication of golden UPF to the designs. The variable can specify a single design name and its corresponding map file, or it can list multiple pairs of design names and corresponding map file names.

For example, to set a single design name and map name:

```
prompt> set upf_name_map [list {my_design my_design.map}]
```

To set multiple designs and corresponding map names:

```
prompt> set upf_name_map [list {top top.map} \
        {mid mid.map}]
```

In a golden UPF session, the tool can change object names as a result of ungrouping and expansion of wildcard names to full names. When the golden UPF and supplemental UPF are applied to a design in a later tool session, the object names used in the golden UPF file must be properly mapped to the new names used in the design netlist. Synopsys Galaxy flow tools all use the same renaming conventions to keep track of changed names.

However, the default renaming conventions cannot handle some complex renaming situations. In those cases, the **write_verilog** command, in addition to writing out the Verilog

netlist, also writes an explicit name mapping file containing the renaming rules for the specific design. It also inserts a link to the name mapping file into the Verilog netlist, in the form of a pragma statement.

In a later tool session, when you read in this Verilog netlist, the tool automatically finds the applicable name map file and properly maps the object names, and it is not necessary to set the **upf_name_map** variable. However, if the name map file has been moved to a new location, or if the Verilog file no longer contains the pragma statement, you can set this variable to specify the location of the name mapping file. The variable setting overrides any conflicting location specified by a pragma in the Verilog netlist file.

The name mapping file is a Tcl script containing Tcl built-in commands and two Synopsys commands: **set_query_rules**, which defines renaming rules for rule-based query, and **define_name_maps**, which defines the name mapping for specific objects. The file is provided information only; you should not edit or modify it.

The following is an example of a name mapping file:

```
# Query Rules
set_query_rules hierarchical_separator {/ _} bus_notation {[] __}

# Tcl built-in: local variables
set hier_1  A/B/C/D
set hier_2  A_B/B2/this/is/a/long/path

# Explicit name maps
define_name_maps \
     -application golden_upf \
     -design_name design \
     -columns {class pattern options names} \
    [list cell $hier_1/A_reg [list leaf]       [list $hier_2/ger_A]
] \
     [list cell A/B/X*_reg    [list nocase leaf] [list A_B/zar_reg A_B/
Y_reg3]]

# cleanup
unset hier_1
unset hier_2
```

To determine the current value of this variable, use **printvar upf_name_map** command. For a list of all mv variables and their current values, use the **print_variable_group mv** command.

SEE ALSO

```
load_upf(2)
save_upf(2)
enable_golden_upf(3)
write_verilog(2)
```

upf_skip_ao_check_for_els_input

Specifies whether enable level shifters inserted by the tool are allowed to use a power supply that is more always-on than the driver supply.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

When this variable is set to **true** (the default), enable level shifters inserted by the tool can use a power supply that is either equally always-on or more always-on than the supply for the driver of the inputs of the enable level shifter.

When this variable is set to **false**, enable level shifters inserted by the tool must use a power supply that is equally always-on compared to the supply for the driver of the inputs of the enable level shifter.

An enable level shifter is a power management cell that performs both isolation and level shifting between two power domains.

SEE ALSO

`insert_mv_cells(2)`

upf_suppress_etm_model_checking

Disables model checking when referring to the UPF of a macro cell.

TYPE

Boolean

DEFAULT

false

GROUP

mv

DESCRIPTION

This variable controls model checking on the UPF for a macro cell. By default, when using the **create_supply_net** and **create_supply_port** commands on a power domain defined at the scope of a macro cell, the tool checks consistency against the cell's power and ground pin definitions. By setting this variable to **true**, these checks are skipped.

SEE ALSO

```
create_supply_net(2)
create_supply_port(2)
```

upf_suppress_message_in_black_box

Suppresses warning messages caused missing objects when UPF commands are loaded at the black box scope.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

When loading UPF at scope of a black box (Verilog stub), netlist objects inside the block are not accessible. If the UPF for the black box contains commands that refer to netlist objects inside the block, those objects are ignored.

When this variable is set to **true** (the default), no warning message is displayed for missing objects. For example,

```
prompt> connect_supply_net sn -port sub/sp
prompt>
```

When this variable is set to **false**, warning messages are displayed for missing objects. For example,

```
prompt> set upf_suppress_message_in_black_box false
false
prompt> connect_supply_net sn -port sub/sp
Warning: Can't find supply ports or power pins matching 'sub/sp' in
design 'mid'. (UID-95)
prompt>
```

SEE ALSO

UPF-623(n)
UPF-626(n)
UPF-627(n)

upf_suppress_message_in_etm

Suppresses warning messages when the current scope is an extracted timing model (ETM) and you use disallowed UPF commands.

TYPE

Boolean

DEFAULT

true

GROUP

none

DESCRIPTION

When loading UPF at the scope of an extracted timing model (ETM) instance, only the **create_power_domain**, **add_port_state** and **set_scope** UPF commands are allowed. Any other UPF commands are ignored.

When this variable is set to **true** (the default), no warning message is issued for ignored commands. For example,

```
prompt> create_supply_net sn
sn
prompt>
```

When this variable is set to **false**, the tool issues a warning message when disallowed commands are ignored. For example,

```
prompt> set_app_var upf_suppress_message_in_etm false
false
prompt> create_supply_net sn
Warning: UPF Command 'create_supply_net' is skipped under ETM scope
'I_SUB'. (UPF-628)
prompt>
```

SEE ALSO

UPF-628(n)

use_improved_icdb

Enables interclock delay balancing to use the delay calculation model specified by the **set_delay_calculation** command.

TYPE

Boolean

DEFAULT

true

GROUP

cts_variables

DESCRIPTION

This variable enables the **balance_inter_clock_delay** command to honor the delay calculation model specified by the **set_delay_calculation** command.

When set to **true**, the **balance_inter_clock_delay** command uses the Arnoldi delay calculation model if you previously ran the **set_delay_calculation** command with the **-arnoldi** or **-clock_arnoldi** options; otherwise it uses the Elmore delay calculation model.

When set to **false** (the default), the **balance_inter_clock_delay** command uses the Elmore delay calculation model regardless of the delay calculation model set with the **set_delay_calculation** command.

SEE ALSO

`balance_inter_clock_delay(2)`

use_port_name_for_oscs

Specifies that when off-sheet connectors for nets also have ports on them, they are given the name of the port.

TYPE

Boolean

DEFAULT

false

GROUP

schematic_variables

DESCRIPTION

This variable specifies that when off-sheet connectors for nets also have ports on them, they are given the name of the port.

By default, the connectors are given the name of the net.

SEE ALSO

vao_feedthrough_module_name_prefix

Variable to specify the prefix for the name of new hierarchies created during voltage area aware always-on synthesis.

TYPE

string

DEFAULT

""

GROUP

mv

DESCRIPTION

In the voltage area aware always-on synthesis triggered by the **psynopt** or the **place_opt** command, new hierarchies are created to hold the newly inserted always-on buffers, to fix DRC violations. Using this variable you can specify prefix for the new hierarchy created during voltage area aware always-on synthesis.

Multicorner-Multimode Support

This variable has no dependency on scenario-specific information.

EXAMPLE

For the new hierarchies to start with the prefix SNPS_VAO, set the variable as follows:

```
set_app_var vao_feedthrough_module_name_prefix {SNPS_VAO}
```

SEE ALSO

```
psynopt(2)
place_opt(2)
```

verbose_messages

Causes more explicit system messages to be displayed during the current session.

TYPE

Boolean

DEFAULT

true

GROUP

system_variables

DESCRIPTION

This variable causes more explicit system messages to be displayed during the current session.

SEE ALSO

`system_variables(3)`

via_attributes

Contains attributes related to via.

DESCRIPTION

Contains attributes related to via.

You can use the **get_attribute** command to determine value of an attribute, and use the **report_attribute** command to get a report of all attributes on a specified object. List the definitions of attributes using **list_attribute -class via -application**,

Via Attributes

array_size

Specifies the array size of a via object. The format is {**col row**}.

The data type of **array_size** is string.

This attribute is read-only.

bbox

Specifies the bounding box of a via. The **bbox** is represented by a **rectangle**.

The format of a **rectangle** specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding box of a via.

The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ll** of a via by accessing the first element of its **bbox**.

The data type of **bbox_ll** is string.

This attribute is read-only.

bbox_llx

Specifies the x coordinate of the lower-left corner of the bounding box of a via.

The data type of **bbox_llx** is integer.

This attribute is read-only.

bbox_lly

Specifies the y coordinate of the lower-left corner of the bounding box of a via.

The data type of **bbox_lly** is integer.

This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding box of a via.

The **bbox_ur** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **bbox_ur** of a via, by accessing the second element of its **bbox**.

The data type of **bbox_ur** is string.

This attribute is read-only.

bbox_urx

Specifies the x coordinate of the upper-right corner of the bounding box of a via.

The data type of **bbox_urx** is integer.

This attribute is read-only.

bbox_ury

Specifies the y coordinate of the upper-right corner of the bounding box of a via.

The data type of **bbox_ury** is integer.

This attribute is read-only.

cell_id

Specifies the Milkyway design ID in which a via object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

center

Specifies the center position of a via object.

The data type of **center** is string.

This attribute is read-only.

col

Specifies the number of horizontal columns of a via object.

The data type of **col** is integer.

This attribute is read-only.

layer

Specifies the layer name list with which a via object is associated. The format is {**via_layer lower_layer upper_layer**}.

The data type of **layer** is string.

This attribute is read-only.

lower_layer

Specifies the lower layer name.

The data type of **lower_layer** is string.

This attribute is read-only.

lower_layer_double_pattern_mask_constraint

Specifies the coloring information for lower layer of the via. This information is needed in the double patterning flow.

The data type of **lower_layer_double_pattern_mask_constraint** is string.

The following values are valid:

- **any_mask**
- **mask1_soft**
- **mask1_hard**
- **mask2_soft**
- **mask2_hard**
- **same_mask**

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

lower_layer_multiple_pattern_mask_constraint

Specifies the coloring information for lower layer of the via. This information is needed in the multiple patterning flow.

The data type of **lower_layer_multiple_pattern_mask_constraint** is string.

The following values are valid:

any_mask

mask1_soft

mask1_hard

mask2_soft

mask2_hard

mask3_soft

mask3_hard

same_mask

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

name

Specifies the name of a via object.

The data type of **name** is string.

This attribute is read-only.

net_id

Specifies the object ID of the net associated with a via object.

The data type of **net_id** is integer.

This attribute is writable. Use the **set_attribute** command to modify its value on a specified object.

net_type

Specifies the type of net associated with a via object.

The data type of **net_type** is string.

This attribute is read-only.

object_class

Specifies the object class name of a via object, which is **via**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies the object ID in a Milkyway design file.

The data type of **object_id** is integer.

This attribute is read-only.

object_type

Specifies the object type name, which can be **via**, **via_array**, or **via_cell**.

The data type of **object_type** is string.

This attribute is read-only.

orientation

Specifies the orientation of a via object.

The data type of **orientation** is string.

The following values are valid:

N

E

S

W

FN

FE

FS

FW

This attribute is writable. Use the **set_attribute** command to modify its value on a specified object. However, when the object type is via_cell, you cannot change its **orientation**.

owner

Specifies the Milkyway design file name in which a via is located.

The data type of **owner** is string.

This attribute is read-only.

owner_net

Specifies the net name to which a via is connected.

The data type of **owner_net** is string.

This attribute is writable. Use the **set_attribute** command to modify its value on a specified object.

route_type

Specifies the route type of a via.

The data type of **route_type** is string.

The following values are valid:

User Enter or **user_enter**

Signal Route or **signal_route**

Signal Route (Global) or **signal_route_global**

P/G Ring or **pg_ring**

Clk Ring or **clk_ring**

P/G Strap or **pg_strap**

Clk Strap or **clk_strap**

P/G Macro/IO Pin Conn or **pg_macro_io_pin_conn**

P/G Std. Cell Pin Conn or **pg_std_cell_pin_conn**

Zero-Skew Route or **clk_zero_skew_route**

Bus or **bus**

Shield (fix) or **shield**

Shield (dynamic) or **shield_dynamic**

Fill Track or **clk_fill_track**

Unknown or **unknown**

The Tcl variable **mw_attr_value_no_space** determines whether **get_attribute** or **report_attribute** returns route_type containing spaces or underscores.

This attribute is writable. Use the **set_attribute** command to modify its value on a specified object.

row

Specifies the number of vertical rows in a via object.

The data type of **row** is integer.

This attribute is read-only.

upper_layer

Specifies the upper layer name.

The data type of **upper_layer** is string.

This attribute is read-only.

upper_layer_double_pattern_mask_constraint

Specifies the coloring information for upper layer of the via. This information is needed in the double patterning flow.

The data type of **upper_layer_double_pattern_mask_constraint** is string.

The following values are valid:

any_mask

mask1_soft

mask1_hard

mask2_soft

mask2_hard

same_mask

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

upper_layer_multiple_pattern_mask_constraint

Specifies the coloring information for upper layer of the via. This information is needed in the multiple patterning flow.

The data type of **upper_layer_multiple_pattern_mask_constraint** is string.

The following values are valid:

any_mask

mask1_soft

mask1_hard

mask2_soft

mask2_hard

mask3_soft

mask3_hard

same_mask

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

via_layer

Specifies the via layer name.

The data type of **via_layer** is string.

This attribute is writable. Use the **set_attribute** command to modify its value on a specified object.

via_layer_double_pattern_mask_constraint

Specifies the coloring information for via layer of the via. This information is needed in the double patterning flow.

The data type of **via_layer_double_pattern_mask_constraint** is string.

The following values are valid:

any_mask

mask1_soft

mask1_hard

mask2_soft

mask2_hard

same_mask

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

via_layer_multiple_pattern_mask_constraint

Specifies the coloring information for via layer of the via. This information is needed in the multiple patterning flow.

The data type of **via_layer_multiple_pattern_mask_constraint** is string.

The following values are valid:

any_mask

mask1_soft

mask1_hard

mask2_soft

mask2_hard

mask3_soft

mask3_hard

same_mask

This attribute is writable. You can use **set_attribute** to modify its value on a specified object.

via_master

Specifies the via master's name defined in the library's technology file; for via_cell, it is the library cell's name.

The data type of **via_master** is string.

This attribute is writable. Use the **set_attribute** command to modify its value on a specified object.

x_pitch

Specifies the center-to-center spacing of a via object in the horizontal direction.

The data type of **x_pitch** is float.

This attribute is writable. Use the **set_attribute** command to modify its value on a specified object.

y_pitch

Specifies the center-to-center spacing of a via object in the vertical direction.

The data type of **y_pitch** is float.

This attribute is writable. Use the **set_attribute** command to modify its value on a specified object.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
mw_attr_value_no_space(3)
```

via_can_change_pin_shape

Controls the via region shape to guide whether the router can change pin shape.

TYPE

Boolean

DEFAULT

false

GROUP

extract_blockage_pin_via_variables

DESCRIPTION

This variable, when set to **true**, allows the **extract_blockage_pin_via** command to create via regions that extend outside of pins for greater router flexibility, at the cost of more router runtime.

By default, the variable is set to **false**, which causes the **extract_blockage_pin_via** command, at 65 nm technology nodes and below, to create via regions small enough so that when a via is dropped on a pin to make a connection, the lower-level metal enclosure around the pin falls entirely within the metal pin.

Via Regions

In a FRAM view, a via region is an area consisting of one or more rectangular boxes, which may be overlapping, over a pin where the router can make a connection to a port. The via region provides guidance to the router about where to drop a via to make a connection.

By default, the router can use either a wire in the pin layer or a via above the pin layer to make a connection to a pin. In the IC Compiler tool, you can restrict the allowed methods of making connections in each layer by using the following command:

```
prompt> set_route_zrt_common_options \
          -connect_within_pins_by_layer_name \
          { {layer mode} {layer mode} ...}
```

When this option is set for a pin layer, the router attempts to make the lower-metal enclosure fall entirely within the metal pin in the FRAM view. This "within-pin" connection ensures that the router will not introduce any DRC violations in the lower metal layer of the cell being connected.

However, if the router is unable to fit the lower-metal enclosure within the pin, it makes the connection anyway, causing the lower-metal enclosure to extend beyond the pin shape. This type of connection is said to "change the pin shape." In that case, the router must spend time checking for and fixing lower-metal design rule violations in the cell.

The manner of via region generation depends on the technology geometry size. For technology nodes at 90 nm and above, the blockage, pin, and via extraction process allows full flexibility to the router without considering whether the pin shape needs to be changed. For technology nodes at 65 nm and below, by default, the blockage, pin, and via extraction process creates more restrictive via areas that ensure "within-pin" via connections that do not change the metal pin shape, so that routing can be performed more quickly.

Enforcing all "within-pin" connections can be too restrictive in certain situations, such as when pin is small and the minimum enclosure around the via is large. In that case, you can override the default behavior of the **extract_blockage_pin_via** command by setting the **via_can_change_pin_shape** variable to **true**.

This allows the **extract_blockage_pin_via** command to create via regions that extend outside of pins for greater router flexibility, at the cost of more router runtime.

If pins are smaller than the minimum via size, then no via regions are generated at all by default. To allow via regions to be generated under these conditions, set the **generate_via_region_when_pin_width_all_less_than_via_width** variable to **true**. This causes the **extract_blockage_pin_via** command to generate via regions for pins, even when the width of the whole pin in smaller than the via width.

The IC Compiler tool lets you edit via regions in FRAM views so that you can control the behavior of the router in making connections to ports through vias. Use the commands **create_via_region**, **write_via_region** and **remove_via_region**.

SEE ALSO

```
extract_blockage_pin_via(2)
generate_via_region_when_pin_width_all_less_than_via_width(3)
```

via_master_attributes

Contains attributes related to via master.

DESCRIPTION

Contains attributes related to via master.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified via master. Specified with **list_attribute -class via_master -application**, the definition of attributes can be listed.

Via Master Attributes

contactcode_number

Specifies the contact code number of the specified via master.

This attribute is read-only. And its data type is integer.

definition_type

Specifies the definition type of a via master. This attribute is read-only. Its data type is string.

Legal values: lib_parameter, lib_rectangle, design_parameter, design_rectangle.

The value **lib_parameter** means that a via master is defined in Milkyway library file and it is defined by parameters such as lower_layer_enc_height, etc. For example, we can find a via master whose **definition_type** is **lib_parameter** in Technology File as:

```
ContactCode"VIA1_VV_11" {
contactCodeNumber= 1
cutLayer= "CUT12"
lowerLayer= "MET1"
upperLayer= "MET2"
isFatContact= 1
cutWidth= 0.1
cutHeight= 0.1
upperLayerEncWidth= 0
upperLayerEncHeight= 0.04
lowerLayerEncWidth= 0
lowerLayerEncHeight= 0.04
minCutSpacing= 0.1
}
```

The value **lib_rectangle** means that a via master is defined in Milkyway library file and it is defined by rectangles. For example, we can find a via master whose **definition_type** is **lib_rectangle** in Technology File as:

```
ContactCode"VIA1_HV" {
contactCodeNumber= 7
cutLayer= "CUT12"
lowerLayer= "MET1"
upperLayer= "MET2"
upperLayerRectTbl= ("0, -0.2, 0.7, 0.2")
lowerLayerRectTbl= ("0, -0.2, 0.7, 0.2")
cutLayerRectTbl = ("0.55, -0.05, 0.65, 0.05",
   "0.05, -0.05, 0.15, 0.05")
}
```

The value **design_parameter** means that a via master is defined in Milkyway design file and it is defined by parameters such as lower_layer_enc_height, etc. Please refer to **create_via_master** for more information.

The value **design_rectangle** means that a via master is defined in Milkyway design file and it is defined by rectangles. Please refer to **create_via_master** for more information.

lower_layer

Specifies the lower layer name of a via master. This attribute is read-only. Its data type is string.

lower_layer_enc_height

Specifies the lower metal layer enclosure height of a via master, in microns. Only exists on via master with **lib_parameter** or **design_parameter** definition_type. This attribute is read-only. Its data type is double.

lower_layer_enc_width

Specifies the lower metal layer enclosure width of a via master, in microns. Only exists on via master with **lib_parameter** or **design_parameter** definition_type. This attribute is read-only. Its data type is double.

min_cut_spacing

Specifies the minimum spacing between cuts when via master is used to form an array, in microns. Only exists on via master with **lib_parameter** or **design_parameter** definition_type.

This attribute is read-only. Its data type is double.

name

Specifies the name of a via master. This attribute is read-only. Its data type is string.

object_class

Specifies object class name. Always be **via_master**. This attribute is read-only. Its data type is string.

rectangles

Specifies the list of rectangles describing the geometry of a via master. Only exists on via master with **lib_rectangle** or **design_rectangle** definition_type. The list are formatted as " {lower_layer_name lower_layer_rectangle1 .. lower_layer_rectangleL} {via_layer_name via_layer_rectangle1 .. via_layer_rectangleN} {upper_layer_name upper_layer_rectangle1 ... upper_layer_rectangleM}"

This attribute is read-only. Its data type is string.

upper_layer

Specifies the upper layer name of a via master. This attribute is read-only. It's data type is string.

upper_layer_enc_height

Specifies the upper metal layer enclosure height of a via master, in microns. Only exists on via master with **lib_parameter** or **design_parameter** definition_type. This attribute is read-only. Its data type is double.

upper_layer_enc_width

Specifies the upper metal layer enclosure width of a via master, in microns. Only exists on via master with **lib_parameter** or **design_parameter** definition_type. This attribute is read-only. Its data type is double.

via_layer

Specifies the cut layer name of a via master. This attribute is read-only. Its data type is string.

via_layer_height

Specifies the cut height of a via master, in microns. Only exists on via master with **lib_parameter** or **design_parameter** definition_type. This attribute is read-only. Its data type is double.

via_layer_width

Specifies the cut width of a via master, in microns. Only exists on via master with **lib_parameter** or **design_parameter** definition_type. This attribute is read-only. Its data type is double.

excluded_for_signal_route

Specifies the design via master can be used by router or not. And its data type is boolean.

SEE ALSO

```
create_via_master(2)
get_attribute(2)
list_attribute(2)
report_attribute(2)
```

via_region_attributes

Contains attributes related to via region.

DESCRIPTION

Contains attributes related to via region.

You can use **get_attribute** to determine value of an attribute, and use **report_attribute** to get a report of all attributes on specified object. Specified with **list_attribute -class via_region -application**, the definition of attributes can be listed.

Via Region Attributes

bbox

Specifies the bounding-box of a via region. The **bbox** is represented by a **rectangle**.

The format of a *rectangle* specification is {{llx lly} {urx ury}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding-box of a via region.

The **bbox_ll** is represented by a **point**. The format of a *point* specification is {x y}.

You can get the **attr_name** of a via region, by accessing the first element of its **bbox**.

The data type of **bbox_ll** is string.

This attribute is read-only.

bbox_llx

Specifies x coordinate of the lower-left corner of the bounding-box of a via region.

The data type of **bbox_llx** is double.

This attribute is read-only.

bbox_lly

Specifies y coordinate of the lower-left corner of the bounding-box of a via region.

The data type of **bbox_lly** is double.

This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding-box of a via region.

The **bbox_ur** attribute is represented by a point. The format of a point specification is {x y}.

You can get the **bbox_ur** of a via region, by accessing the second element of its **bbox**.

The data type of **bbox_ur** is string.

This attribute is read-only.

bbox_urx

Specifies x coordinate of the upper-right corner of the bounding-box of a via region.

The data type of **bbox_urx** is double.

This attribute is read-only.

bbox_ury

Specifies y coordinate of the upper-right corner of the bounding-box of a via region.

The data type of **bbox_ury** is double.

This attribute is read-only.

cell_id

Specifies Milkyway design ID in which a via region object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

name

Specifies name of a via region object.

The data type of **name** is string.

This attribute is read-only.

number_of_points

Specifies the number of points to illustrate the boundary of a via region object.

The data type of **number_of_points** is integer.

You can refer to the attribute **points**. The list length of **points** is the value of **number_of_points**.

This attribute is read-only.

object_class

Specifies object class name of a via region, which is **via_region**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies object ID in Milkyway design file.

The data type of **object_id** is integer.

This attribute is read-only.

owner_port

Specifies port name which a via region object is associated with.

The data type of **owner_port** is string.

This attribute is read-only.

points

Specifies points of the boundary of a via region. A via region can be a rectangle, a rectilinear polygon, or multiple rectangles.

When a via region is either a rectangle or a rectilinear polygon, its **points** is represented by a list of points. The last element of the list is the same as the first element.

When a via region consists of multiple rectangles, its **points** is represented by a list of points of rectangles. Every five points represent one rectangle.

The data type of **points** is string.

This attribute is read-only.

is_rotate90

Specifies whether a via region is used for the contactCode object who is rotated 90 degree.

The data type of **is_rotate90** is boolean.

This attribute is read-only.

via_master

Specifies the name of the via master associated with a via region object.

The data type of **via_master** is string.

This attribute is read-only.

SEE ALSO

```
get_attribute(2)
list_attributes(2)
report_attribute(2)
```

view_analyze_file_suffix

Specifies, in a list of file extensions, the files shown in the File/Analyze dialog box of Design Analyzer.

TYPE

string

DEFAULT

v vhd vhdl

GROUP

suffix_variables

DESCRIPTION

Specifies, in a list of file extensions, the files shown in the File/Analyze dialog box of Design Analyzer. The default value is {*v*, *vhd*, *vhdl*}.

To determine the current value of this variable, type **printvar view_analyze_file_suffix**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_arch_types

Sets the contents of the architecture option menu. Contains a list of host machine architectures you can use for background jobs from the Design Analyzer viewer.

TYPE

list

DEFAULT

sparcOS5 hpux10 rs6000 sgimips

GROUP

view_variables

DESCRIPTION

This variable sets the contents of the architecture option menu. It contains a list of host machine architectures you can use for background jobs from the Design Analyzer viewer.

SEE ALSO

view_background

Specifies the background color of the Design Analyzer viewer.

TYPE

string

DEFAULT

black

GROUP

view_variables

DESCRIPTION

Specifies the background color of the Design Analyzer viewer. Valid settings are *white* (the default) and *black*.

To determine the current value of this variable, type **printvar view_background**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_cache_images

Specifies to Design Analyzer that the tool is to cache bitmaps for fast schematic drawing.

TYPE

string

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Specifies that the tool is to cache bitmaps for fast schematic drawing. The default is *true*.

To determine the current value of this variable, type **printvar view_cache_images**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_command_log_file

Names a file and its location that is to contain all text written to the Design Analyzer Command window.

TYPE

string

DEFAULT

"./view_command.log"

GROUP

view_variables

DESCRIPTION

Names a file and its location that is to contain all text written to the Design Analyzer Command window. The default is to set this variable.

To determine the current value of this variable, use **printvar view_command_log_file**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

```
command_log_file(3)
view_log_file(3)
```

view_command_win_max_lines

Contains the maximum number of lines to be saved in the Design Analyzer command window.

TYPE

integer

DEFAULT

1000

GROUP

view_variables

DESCRIPTION

Contains the maximum number of lines to be saved in the Design Analyzer command window. When the number of lines of output added to the command window exceed the number this variable specifies, the older lines at the top of the list are removed.

This variable should be set to 1000 or more. Values up to tens of thousands are also useful.

To determine the current value of this variable, type **printvar view_command_win_max_lines**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_dialogs_modal

Requires that the question and error dialogs in Design Analyzer be confirmed, before you can continue entering commands.

TYPE

Boolean

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Requires that the question and error dialogs in Design Analyzer be confirmed, before you can continue entering commands.

To determine the current value of this variable, use **printvar view_dialogs_modal**. For a list of all **view** variables and their current values, type the **print_variable_group view**.

SEE ALSO

view_disable_cursor_warping

Causes the cursor to be automatically "warped" (moved). When *false*, the cursor is automatically "warped" (or moved) to dialog boxes.

TYPE

Boolean

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Causes the cursor to be automatically "warped" (moved). When *false*, the cursor is automatically "warped" (or moved) to dialog boxes. The default is *true*.

To determine the current value of this variable, use **printvar view_disable_cursor_warping**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_disable_error_windows

Instructs Design Analyzer not to post the error windows when errors occur.

TYPE

Boolean

DEFAULT

false

GROUP

view_variables

DESCRIPTION

Instructs Design Analyzer not to post the error windows when errors occur. The default is *false*.

To determine the current value of this variable, type **printvar view_disable_error_windows**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_disable_output

Disables output to the Design Analyzer command window.

TYPE

Boolean

DEFAULT

false

GROUP

view_variables

DESCRIPTION

Disables output to the Design Analyzer command window, when set to *true*. This variable is useful when you run Design Analyzer over slow networks, such as telephone lines. The default value is *false*.

To determine the current value of this variable, type **printvar view_disable_output**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_error_window_count

Specifies the maximum number of errors Design Analyzer reports for a command.

TYPE

integer

DEFAULT

6

GROUP

view_variables

DESCRIPTION

Specifies the maximum number of errors Design Analyzer reports for a command. The default value is *6*. If more than the specified number of errors occurs, you are informed that you can see additional errors in the command window. The error window is suppressed until the end of the command.

To display all errors, set this variable to 0. To display no errors, set this variable to a negative number or set the **view_disable_error_windows** variable to *true*.

To determine the current value of this variable, type **printvar view_error_window_count**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

`view_disable_error_windows(3)`

view_execute_script_suffix

Displays only files with the stated suffixes, from directories you select in the Execute Script option window of the Setup menu of Design Analyzer.

TYPE

list

DEFAULT

".script .scr .dcs .dcv .dc .con .tcl"

GROUP

suffix_variables

DESCRIPTION

Displays only files with the stated suffixes, from directories you select in the Execute Script option window of the Setup menu of Design Analyzer.

To determine the current value of this variable, type **printvar view_execute_script_suffix**. For a list of all **suffix** variables and their current values, type **print_variable_group suffix**.

SEE ALSO

view_info_search_cmd

Invokes, if set, the online information viewer through the optional menu item On-Line Information.

TYPE

string

DEFAULT

""

GROUP

view_variables

DESCRIPTION

Invokes, if set, the online information viewer through the optional menu item On-Line Information. Set the value of this variable to the UNIX path name to the online information viewer.

To determine the current value of this variable, type **printvar view_info_search_cmd**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_log_file

Specifies the file in which the tool stores events that occur in the viewer.

TYPE

string

DEFAULT

""

GROUP

view_variables

DESCRIPTION

Specifies the file in which the tool stores events that occur in the viewer. This variable is useful for error reporting. You can execute this variable with the Execute Script option of the Setup menu, or insert the file, using the **include** command in Design Analyzer.

To determine the current value of this variable, type **printvar view_log_file**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_on_line_doc_cmd

Invokes, if set, the online documentation viewer, through the optional menu item On-Line Documentation.

TYPE

string

DEFAULT

""

GROUP

view_variables

DESCRIPTION

Invokes, if set, the online documentation viewer, through the optional menu item On-Line Documentation. Set the value of this variable to the UNIX command to invoke the online documentation view: for example, /vobs/snps/synopsys/sold.

To determine the current value of this variable, type **printvar view_on_line_doc_cmd**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_read_file_suffix

Displays only files with the stated suffixes, from directories you select with the Read option of the File menu of Design Analyzer.

TYPE

list

DEFAULT

db gdb sdb edif eqn fnc lsi mif NET pla st tdl v vhd vhdl xnf

GROUP

suffix_variables

DESCRIPTION

Displays only files with the stated suffixes, from directories you select with the Read option of the File menu of Design Analyzer.

To determine the current value of this variable, type **printvar view_read_file_suffix**. For a list of all **suffix** variables and their current values, type **print_variable_group suffix**.

SEE ALSO

view_script_submenu_items

Allows users to add to the Design Analyzer Setup pulldown menu valid items to invoke user scripts.

TYPE

string

DEFAULT

"DA to SGE Transfer" write_sge

GROUP

view_variables

DESCRIPTION

Allows users to add to the Design Analyzer Setup pulldown menu valid items to invoke user scripts. The variable should contain a list of strings, grouped into pairs. The first member of the pair is the text that will appear in the submenu. The second member is the string that gets sent to the dc_shell command line for execution. You can use any valid dc_shell command sequence.

For example,

view_script_submenu_items = "{"List", "list_instances", "ls", "sh ls -lR", "Update", "include update.dcsh"}"

creates a submenu under the Scripts menu item on the Setup pulldown menu. The submenu contains the strings List, ls, and Update. Selecting one of these entries executes the commands **list_instances**, **sh ls -lR**, or **include update.dcsh**, respectively.

The tool reads this variable only at startup time, so any changes after the Design Analyzer is initialized are not reflected.

To determine the current value of this variable, type **list view_script_submenu_items**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_set_selecting_color

Specifies the color to use for selecting and zooming.

TYPE

string

GROUP

view_variables

DESCRIPTION

Specifies the color to use for selecting and zooming. You can set this variable in the **.synopsys_dc.setup** initialization file. Any color in **/usr/lib/X11/rgb.txt** is valid.

To determine the current value of this variable, type **printvar view_set_selecting_color**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_tools_menu_items

Permits partial configuration of the Tools pulldown menu to add a new menu item for invoking user scripts.

TYPE

string

DEFAULT

""

GROUP

view_variables

DESCRIPTION

Permits partial configuration of the Tools pulldown menu to add a new menu item for invoking user scripts. This .synopsys_dc.setup file variable contains a list of strings, grouped into pairs. The first member of the pair is the text that appears in the submenu. The second member is the string that is sent to the dc_shell command line for execution. You can use any valid dc_shell command sequence.

For example,

view_tools_menu_items = "{"List", "list_instances", "ls", "sh ls -lR", "Update", "include update.dcsh"}"

adds three more items to the Tools menu: List, ls, and Update. Selecting one of these entries executes one of the commands. For instance, if you selected **update**, the **include update.dcsh** command would be executed.

The tool reads this variable only at start-up time. Any changes after the Design Analyzer is initialized are not reflected.

To determine the current value of this variable, type **printvar view_tools_menu_items**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_use_small_cursor

Specifies to the tool that the X display is to support only 16 x 16-bit map size cursors.

TYPE

string

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Specifies to the tool that the X display is to support only 16 x 16-bit map size cursors. To achieve this result, set the value of this variable to *true*. The default is *""*.

To determine the current value of this variable, type **printvar view_use_small_cursor**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_use_x_routines

Enables the use of internal arc-drawing routines (instead of X routines).

TYPE

Boolean

DEFAULT

true

GROUP

view_variables

DESCRIPTION

Enables the use of internal arc-drawing routines (instead of X routines). If there is a math coprocessor chip on the same machine that the X server is on, X arc-drawing routines are faster. Otherwise, internal arc-drawing routines are faster. The default value is *true*.

To determine the current value of this variable, type **printvar view_use_x_routines**. For a list of all **view** variables and their current values, type **print_variable_group view**.

SEE ALSO

view_write_file_suffix

Displays only files with the stated suffixes, from directories you select with the Save As option of the File menu of Design Analyzer.

TYPE

list

DEFAULT

gdb db sdb do edif eqn fnc lsi NET neted pla st tdl v vhd vhdl xnf

GROUP

suffix_variables

DESCRIPTION

Displays only files with the stated suffixes, from directories you select with the Save As option of the File menu of Design Analyzer.

To determine the current value of this variable, type **printvar view_write_file_suffix**. For a list of all **suffix** variables and their current values, type **print_variable_group suffix**.

SEE ALSO

voltage_area_attributes

DESCRIPTION

This man page describes the attributes related to voltage areas.

You can use the **get_attribute** command to determine the value of an attribute and the **report_attribute** command to get a report of all the attributes on a specified object.

To see a list of all voltage area attributes, use the **list_attributes -application -class voltage_area** command.

Voltage Area Attributes

bbox

Specifies the bounding box of a voltage area.

The bounding box is represented by a rectangle. The format of a rectangle specification is {{*llx lly*} {*urx ury*}}, which specifies the lower-left and upper-right corners of the rectangle.

The data type of **bbox** is string.

This attribute is read-only.

bbox_ll

Specifies the lower-left corner of the bounding box of a voltage area.

The **bbox_ll** is represented by a point. The format of a point specification is {*x y*}.

You can get the **bbox_ll** of a voltage area by accessing the first element of its **bbox**.

The data type of **bbox_ll** is string.

This attribute is read-only.

bbox_llx

Specifies the x-coordinate of the lower-left corner of the bounding box of a voltage area.

The data type of **bbox_llx** is double.

This attribute is read-only.

bbox_lly

Specifies the y-coordinate of the lower-left corner of the bounding box of a voltage area.

The data type of **bbox_lly** is double.

This attribute is read-only.

bbox_ur

Specifies the upper-right corner of the bounding box of a voltage area.

The **box_ur** is represented by a point. The format of a point specification is {*x y*}.

You can get the **bbox_ur** of a voltage area by accessing the second element of its **bbox**.

The data type of **bbox_ur** is string.

This attribute is read-only.

bbox_urx

Specifies the x-coordinate of the upper-right corner of the bounding box of a voltage area.

The data type of **bbox_urx** is double.

This attribute is read-only.

bbox_ury

Specifies the y-coordinate of the upper-right corner of the bounding box of a voltage area.

The data type of **bbox_ury** is double.

This attribute is read-only.

cell_id

Specifies the identification number of the Milkyway design in which the voltage area object is located.

The data type of **cell_id** is integer.

This attribute is read-only.

color

Specifies the color for a voltage area and its leaf cells.

The data type of **color** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

guardband

Specifies the guardband spacing around a voltage area.

Its format is {*guardband_x guardband_y*}.

The guardband is the spacing along the boundary of a voltage area where cells cannot be placed because of the lack of power supply rails.

The data type of **guardband** is string.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

guardband_x

Specifies the guardband width in the horizontal direction.

The data type of **guardband_x** is float.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

guardband_y

Specifies the guardband width in the vertical direction.

The data type of **guardband_y** is float.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

is_fixed

Specifies whether a voltage area is in a fixed location and therefore ignored during shaping.

The data type of **is_fixed** is Boolean.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

max_buffer_distance

Specifies maximum net length that is allowed to be routed in the particular voltage area. The virtual router and Zroute global router honor it.

The data type of **max_buffer_distance** is float and the unit is microns.

This attribute is writable. You can use the **set_attribute** command to modify its value on a specified object.

modules

Specifies the collection of top node cells inside a voltage area.

The data type of **modules** is collection.

This attribute is read-only.

name

Specifies the name of a voltage area object.

The data type of **name** is string.

This attribute is read-only.

object_class

Specifies the object class name of a voltage area, which is **voltage_area**.

The data type of **object_class** is string.

This attribute is read-only.

object_id

Specifies the object identification number in the Milkyway design database.

The data type of **object_id** is integer.

This attribute is read-only.

points

Specifies the point list of the voltage area's boundary.

The data type of **points** is string.

This attribute is read-only.

utilization

Specifies the utilization of a voltage area, the fraction of the area that is occupied by cells.

The data type of **utilization** is float.

This attribute is read-only.

within_block_abstraction

Specifies whether the voltage area is part of a block abstraction model.

The data type of **within_block_abstraction** is Boolean.

This attribute is read-only and cannot be modified.

within_ilm

Specifies whether the voltage area is part of an interface logic model (ILM).

The data type of **within_ilm** is Boolean.

This attribute is read-only and cannot be modified.

SEE ALSO

```
create_voltage_area(2)
get_attribute(2)
list_attributes(2)
report_attribute(2)
set_attribute(2)
```

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_multibits
get_nets
get_pins
get_ports
get_references
list_designs
list_instances
list_libs
trace_nets
untrace_nets
```

In addition to these commands, commands that perform an implicit get also support the wildcarding feature.

Escaping Wildcards

Wildcard characters must be escaped using double backslashes (\\) to remove their special regular expression meaning. See the EXAMPLES section for more information.

Escaping Escape Character (\\)

This is similar to that of escaping wildcard characters, but needs one escape character each to escape the escape character. See the EXAMPLES section for more information.

EXAMPLES

The following are examples of using 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:

```
prompt> 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:

```
prompt> get_cells U*
{"U1" "U2" "U3" "U4"}
```

Escaping Wildcards

The following are examples of escaping wildcard characters.

The following example gets design test?1 in the system.

```
prompt> get_designs {test\\?1}
{"test?1"}
```

The same example can be written using the Tcl **list** command:

```
prompt> get_designs [list {test\?1}]
{"test?1"}
```

If neither curly braces nor the **list** command is used, the syntax is as follows:

```
prompt> get_designs test\\\\?1
{"test?1"}
```

Escaping Escape Character (\\)

The following are examples of escaping an escape character.

The following example finds design test\1 in the system.

```
prompt> get_designs {test\\\\1}
{"test\1"}
```

The same above example can written using the Tcl **list** command:

```
prompt> get_designs [list {test\\1}]
{"test\1"}
```

If neither curly braces nor the **list** command is used, the syntax is as follows:

```
prompt> get_designs test\\\\\\\\1
{"test\1"}
```

write_converted_tf_syntax

Controls whether the **write_mw_lib_files** command writes out the converted rule tables when the technology file uses the alternative syntax for area-based fat metal contact rules and fat metal extension contact rules.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls the syntax used by the **write_mw_lib_files** command for the area-based fat metal contact rules and fat metal extension contact rules when these rules are defined in the technology file using the alternative syntax.

The alternative syntax defines these rules by using the enclosureTbl and minCutsTbl tables in the Layer section of the technology file. Internally, the tool converts these rules to ContactCode sections and via rules in the Layer section.

By default, the **write_mw_lib_files** command writes these rules using the syntax from the input technology file. If you set this variable to **true**, the **write_mw_lib_files** command writes these rules using the internally-derived syntax (the Contact Code sections and via rules in the Layer sections).

For details about the technology file syntax for these rules, see the *IC Compiler Technology File and Routing Rules Reference Manual*.

SEE ALSO

`write_mw_lib_files(2)`

write_name_nets_same_as_ports

Specifies to the tool that nets are to receive the same names as the ports to which the nets are connected.

TYPE

Boolean

DEFAULT

false

GROUP

edif_variables, io_variables

DESCRIPTION

This variable specifies to the tool that nets are to receive the same names as the ports to which the nets are connected. This variable affects nets in design descriptions written in EDIF, LSI, or TDL format. (Other nets might be renamed to avoid creating shorts.)

Note that in the EDIF format, even if the **edifout_power_and_ground_representation** variable is set to **port**, the power and ground nets will not be named the same as the power and ground ports that are written in the interface construct of each cell construct.

Net names in the design database are unchanged; that is, only the file being written out is affected and not the original design data.

SEE ALSO

`io_variables(3)`

write_sdc_output_lumped_net_capacitance

Determines whether or not the **write_sdc** and **write_script** commands output net loads.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable instructs the **write_sdc** and **write_script** commands to output net loads when set to **true**.

When set to **false**, **write_sdc** and **write_script** do not output net loads. The net loads are output through **set_load** command statements during **write_sdc** and **write_script**.

By default all net loads are output during **write_sdc** and **write_script** execution.

SEE ALSO

```
set_load(2)
write_sdc(2)
write_script(2)
```

write_sdc_output_net_resistance

Determines whether or not the **write_sdc** and **write_script** commands output net resistance.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable instructs the **write_sdc** and **write_script** commands to output net resistance. When set to false, **write_sdc** and **write_script** do not output net resistance. The net resistances are output through **set_resistance** statements during **write_sdc** and **write_script**.

By default, all net resistances are output during **write_sdc** and **write_script** execution.

SEE ALSO

```
set_resistance(2)
write_sdc(2)
write_script(2)
```

xt_filter_logic_constant_aggressors

Affects the behavior of the **report_noise**, **report_timing**, and **compile** commands with crosstalk or noise effect enabled. Specifies if logic constant nets should be considered as aggressors in crosstalk and static noise analysis and optimization.

TYPE

Boolean

DEFAULT

true

GROUP

si_variables

DESCRIPTION

This variable affects the behavior and correlation of the **report_noise**, **report_timing**, and **compile** commands. By default, logic constant aggressors are not included in crosstalk or static noise analysis and optimization. Aggressor nets can be logic constant because they are connected to tie-off cells or because of case analysis. The **case_analysis_with_logic_constants** timing variable may also determine if an aggressor net is logic constant.

When set to **false**, logic constant aggressors are included in crosstalk or static noise analysis and optimization. To obtain the more accurate behavior of crosstalk timing and static noise analysis, set this variable to **true**.

SEE ALSO

```
report_noise(2)
report_timing(2)
case_analysis_with_logic_constants(3)
si_variables(3)
```

zrt_max_parallel_computations

Sets the maximum degree of parallelism in multicore Zroute.

TYPE

integer

DEFAULT

0

GROUP

Zroute

DESCRIPTION

This application variable specifies the degree of parallelism in multicore Zroute. Increasing parallelism results in reduced runtime.

When set to **0** (the default), the number of cores used for Zroute is specified directly by the value of the **set_host_options** command with the **-max_cores** option value.

When set to **1**, multicore Zroute is disabled.

When the value is smaller than the **-max_cores** value of the **set_host_options** command, the number of cores being used will not exceed this limit.

There is no benefit to setting the value to larger than **-max_cores**. The actual degree of parallelism (number of processes/threads) will not exceed the value specified here, but may not always be exactly the same. The tool derives the number internally to achieve best performance.

SEE ALSO

`set_host_options(2)`