Global Variable Reference for BuildGates® Synthesis and Cadence® PKS

Product Version 5.0.13
December 2003

Cadence Design Systems, Inc., River Oaks Parkway, San Jose, CA 34, USA

Contents

8
Name Changing Globals

12
Temporary Global Variables

Preface

This preface contains the following sections:

About This Manual

This manual is a complete listing of the BuildGates Global Commands.

Other Information Sources

For more information about related products, you can consult the sources listed here. The documents you have will vary depending on your product licenses.

- *AmbitWare Component Reference*
- *BuildGates Synthesis User Guide*
- *CeltIC User Guide*
- *Command Reference for BuildGates Synthesis and Cadence PKS*
- *Common Timing Engine (CTE) User Guide*
- *Constraint Translation for BuildGates Synthesis and Cadence PKS*
- *Datapath for BuildGates Synthesis and Cadence PKS*
- *Delay Calculation Algorithm Guide*
- *Design for Test Using BuildGates Synthesis and Cadence PKS*
- *Distributed Processing for BuildGates Synthesis*
- *Glossary for BuildGates Synthesis and Cadence PKS*
- *GUI Guide for BuildGates Synthesis and Cadence PKS*

- *HDL Modeling for BuildGates Synthesis*
- *Low Power for BuildGates Synthesis and Cadence PKS*
- *Low Power Synthesis Tutorial*
- *Migration Guide for BuildGates Synthesis and Cadence PKS*
- *Modeling Generation for Verilog 1 and the Verilog Datapath Extension*
- *PKS User Guide*
- *Synthesis Place-and-Route Flow Guide*
- *Verilog Datapath Extension Reference*
- *VHDL Datapath Package Reference*
- *Known Problems and Solutions in BuildGates Synthesis*
- *Know Problems and Solutions in Cadence PKS*
- *What's New in Cadence PKS*
- *What's New in BuildGates Synthesis*

BuildGates synthesis is often used with other Cadence® tools during various design flows. The following documents provide information about these tools and flows. Availability of these documents depends on the product licenses your site has purchased.

- *Cadence Timing Library Format Reference*
- *Cadence Pearl Timing Analyzer User Guide*
- *Cadence General Constraint Format Reference*

The following books are helpful references, but are not included with the CD-ROM documentation:

- IEEE 4 Verilog HDL LRM
- TCL Reference, *Tcl and the Tk Toolkit*, John K. Ousterhout, Addison-Wesley Publishing Company

Documentation Conventions

Text Command Syntax

The list below describes the syntax conventions used for the BuildGates Synthesis text interface commands.

`literal`	Nonitalic words indicate keywords that you must enter literally. These keywords represent command or option names.
`argument`	Words in italics indicate user-defined arguments or information for which you must substitute a name or a value.
|	Vertical bars (OR-bars) separate possible choices for a single argument.
`[ ]`	Brackets denote optional arguments. When used with OR-bars, they enclose a list of choices from which you can choose one.
`{ }`	Braces are used to indicate that a choice is required from the list of arguments separated by OR-bars. You must choose one from the list. `{ argument1 | argument2 | argument3 }`
`...`	Three dots (...) indicate that you can repeat the previous argument. If the three dots are used with brackets (that is, `[argument]...`), you can specify zero or more arguments. If the three dots are used without brackets (`argument...`), you must specify at least one argument, but can specify more.
#	The pound sign precedes comments in command files.

Using Menus

GUI commands can take one of three forms.

CommandName	A command name with no dots or arrow executes immediately.
CommandName…	A command name with three dots displays a form for choosing options.

CommandName ->	A command name with a right arrow displays an additional menu with more commands. Multiple layers of menus and commands are presented in what are called command sequences, for example: *File – Import – LEF*. In this example, you go to the File menu, then the Import submenu, and, finally, the LEF command.

Using Forms

…	A menu button that contains only three dots provides browsing capability. When you select the browse button, a list of choices appears.
Ok	The *Ok* button executes the command and closes the form.
Cancel	The *Cancel* button cancels the command and closes the form.
Defaults	The *Defaults* button displays default values for options on the form.
Apply	The *Apply* button executes the command but does not close the form.

1

Introduction

This document is organized into the following chapters. The globals are grouped according to Table 1-1 on page 24. A list of all globals can be found in the Index on page 193.

- **Chapter 2, “Common Globals”**
- Chapter 3, “Ambitware Globals”
- **Chapter 4, “CTPKS Globals”**
- **Chapter 5, “Distributed Synthesis Globals”**
- **Chapter 6, “HDL Globals”**
- **Chapter 7, “Low Power Synthesis Globals”**
- Chapter 8, “Name Changing Globals”
- **Chapter 9, “PKS Globals”**
- **Chapter 10, “Test Synthesis Globals”**
- **Chapter 11, “Timing Globals”**

This document contains the BuildGates Synthesis global `ac_shell` variables, including the default values for the arguments and the functionality of each argument.

For each of these globals, you can retrieve the current setting using the command `get_global` *`global_name`*. For example:

```
get_global hdl_error_on_latch
```

The `reset_global` *`global_name`* command returns the setting to the default value. For example, the following command changes the current setting back to the default value of `false`:

```
reset_global hdl_error_on_latch
```

Table 1-1 Global Command Groups

Global Group	Description	Prefix*
Common	Common global commands	no prefix
AMBITWARE	Ambitware	`aware`
CTPKS	Clock Tree Physically Knowledgeable Synthesis	`ctpks_`
DCN	Do Change Name global commands	`dcn_`
DFT	Design For Test Synthesis	`dft_`
DIST	Distributed Synthesis	`dist_`

HDL	Hardware Definition Languages	`edifin_`
		`edifout_`
		`hdl_`

PKS	Physically Knowledgeable Synthesis	`pks_`
POWER	Power Synthesis	`power_`
TIMING	Timing Synthesis	`various`

* Some commands in the grouping may not have the designated prefix.

2

Common Globals

acsh_prompt

`acsh_prompt` *`string`*

Specifies the command prompt string for `ac_shell`.

Default: `[%h]`

add_tech_data_from_other_lef_files

`add_tech_data_from_other_lef_files {true | false}`

Includes technology data from secondary LEF files.Technology data from the primary LEF file is read in using the `read_lef` command. When the global `add_tech_data_from_other_lef_files` is set to `true`, technology data present in LEF files being read in using the `read_lef_update` command are also added to the primary tech library.

Default: `false`

auto_line_length

`auto_line_length {true | false}`

When set to `true`, dynamically changes the `line_length` global to be the width of the terminal window.

Default: `false`

auto_slew_prop_selection

`auto_slew_prop_selection {true | false}`

When set to `false`, all optimizations are performed in fast slew propagation mode using the depth specified in the global `depth_for_fast_slew_prop`.

When set to `true`, all optimizations are performed in worst slew propagation mode using a depth increasing from 0 to 1 to 2. When slack reaches 0 or is no longer improving, the tool increases the slew depth to 1 and repeats the process. After optimization is run with slew depth 2 the tool is finished.

If the `-incremental` option of a `do_xform_*` command is specified, optimization will always start with depth of 2.

Default: `true`

bidi_io_arc

```
bidi_io_arc {enable | disable}
```

Enables the bidirectional feedback paths in a cell by establishing a feedback path(s) between the cell pin(s) feeding the bidi port and the driven cell pin(s). This variable has no effect on timing of bidirectional feedback paths involving more than one cell.

Note: This global has been renamed: `timing_disable_internal_inout_cell_paths`, which has the opposite meaning.The default behavior of the software is still the same.

Default: `disable`

build_id

```
build_id
```

Returns the id number used to reference which particular build of a given release is being used; similar to the version number. For example: `get_global version` returns a version number such as v4.0-s009 (Jul 1 2001 03:56:10) and `get_global build_id` returns a build id such as v04.00-s009+50.

buscomp_generator

```
buscomp_generator string
```

Choose a scheme to name individual bits of buses. The *`string`* argument must include `%s` to indicate the name of the bus signal, and `%d` to indicate the bit number. This global must be set before `do_build_generic` is executed. Related commands are `do_blast_busses` and `do_rename`.

Default: `%s[%d]`

capacitance_limit

`capacitance_limit` *`float`*

This variable has been replaced with the variable `max_capacitance_limit`.

cmdfile

`cmdfile` *`filename`*

Sets the name of the command file.

Default: `ac_shell.cmd`

congestion_scale_factor

`congestion_scale_factor` *`float`*

Multiplies the routing supply (tracks available for routing) during routing congestion analysis. Makes a design appear more (value < 1.0) or less (value > 1.0) congested than it actually is. If this global is set to 1.0, then congestion is not scaled.

Default: `1.0`

const_output_max_cap

`const_output_max_cap` *`float`*

Determines the maximum output capacitance a constant cell can drive. If the value is more restrictive than the limit present in the library, this will override the library limit.

Default: infinity

const_output_max_fanout

`const_output_max_fanout` *`float`*

Determines the maximum fanouts a constant cell can drive. If the value is more restrictive than the limit present in the library, this will override the library limit.

Default: infinity

crosstalk_threshold

```
crosstalk_threshold
```

Specifies a new threshold against which crosstalk prevention is performed.

Default: `MAXFLOAT`

crosstalk_tolerance

```
crosstalk_tolerance
```

Specifies a number, which together with the calculated or specified threshold, defines a range within which crosstalk violations will be tolerated and need not be fixed. Thus, the crosstalk violation at a net is calculated as follows:

```
violation = worst net transition time - (threshold + tolerance)
```

where a positive number indicates a violation.

Default: `0`

depth_for_fast_slew_prop

```
depth_for_fast_slew_prop { 0 | 1 | 2 | 3 }
```

Controls the number of logic levels that input slew affects output slew when using fast slew propagation mode. Increasing the depth converges towards worst slew propagation mode timing. See also `slew_propagation_mode` global.

Default: `0`

dissolve_naming_style

```
dissolve_naming_style {none | hierarchical}
```

Controls instance names after dissolving modules.

When set to `hierarchical`, all instances coming from dissolved modules will have their parent instance name prefixed to their name.

Default: `none`

For example, consider the following module hierarchy where `top` instantiates `mid` (instance name `m1`) and `mid` instantiates `bot1` (instance name `b1`) and `bot2` (instance name `b2`).

```
top----
        |
        |
        m1 (mid)-----------------
                   |            |
                   |            |
                   b1 (bot1)    b2 (bot2)
```

If this global is set to `hierarchical` after you issue the `do_dissolve_hierarchy` command, the name of instances in module `top` would be:

Original instance name	Instance name after dissolving
`b1`	`m1_b1`
`b2`	`m1_b2`

When the global is set to `none`, the instances retain their original name after dissolving, except when there is a name collision.

Important

Any `do_xform` command issued with this global set to `hierarchical` would create long instance names. Therefore, reset the global to `none` before issuing the `do_xform` command.

do_optimize_skew_clock_buffer_list

`do_optimize_skew_clock_buffer_list` *`list_of_buffers`*

When set to an non-empty list of buffers, instructs the `do_optimize -skew_clock` command to use only buffers from this list.

Note: The specified buffers must be in a target technology library.

Default: `{}`

do_optimize_skew_clock_ignore_dont_utilize

`do_optimize_skew_clock_ignore_dont_utilize {on | off}`

When set to `on`, instructs the `do_optimize -skew_clock` command to try any buffer in the target technology, even if it is marked `dont_utilize`.

Default: `off`

drv_cap_limit_multiplier

`drv_cap_limit_multiplier` *`float`*

Specifies the multiplying factor for the capacitance limits during the design rule fixing and reporting. The specified value affects the number of capacitance violations and the level of each violation. If the specified value is less than `1.0`, it will introduce pessimism.

Default: `1.0`

drv_cap_offset

`drv_cap_offset` *`float`*

Specifies the offset for the max capacitance and min capacitance violations. If the violation values are less than the offset value, then they are neither fixed or reported.

Default: `0.0`

drv_slew_limit_multiplier

`drv_slew_limit_multiplier` *`float`*

Specifies the multiplying factor for the slew limits during the design rule fixing and reporting. The specified value affects the number of slew violations and the level of each violation. If the specified value is less than `1.0`, it will introduce pessimism.

Default: `1.0`

drv_slew_offset

`drv_slew_offset` *`float`*

Specifies the offset for the max transition and min transition violations. If the violation values are less than the offset value, then they will not be fixed or reported.

Default: `0.0`

echo_commands

```
echo_commands {true | false}
```

Echoes each command prior to its execution on standard output.

Default: `false`

enable_auto_congestion_scaling

```
enable_auto_congestion_scaling {true | false}
```

If set to `true`, routing congestion is scaled down at the beginning of each optimization command so that the design always appears uncongested. The scaling is accomplished by automatically computing and setting a value for the global `congestion_scale_factor`. The scale factor is computed only at the beginning of an optimization command; therefore, if a design becomes more congested during optimization, this additional congestion is made visible to PKS.

If set to `false`, the congestion scale factor is not automatically computed.

This global is useful when PKS over estimates congestion, and this over-estimation hinders optimization.

Default: `true`

enable_extra_space_in_congested_bins

```
enable_extra_space_in_congested_bins {true | false}
```

If set to `true`, PKS applies the value assigned to the global `extra_space_for_opt` to congested bins.

If set to `false`, the value assigned to the global `extra_space_for_opt` is not applied to congested bins.

This global is useful in situations where PKS over-estimates congestion, and this over-estimation hinders optimization. Note that for this global to be effective, the value assigned to the global `extra_space_for_opt` must be greater than 0.

Default: `false`

enable_pinswap

```
enable_pinswap {true | false}
```

Enables or disables the pin-swapping algorithm.

Default: `true`

estimate_supply_rail_congestion

```
estimate_supply_rail_congestion {true | false}
```

This variable applies only to designs without rails; it has no effect on designs with rails. When an optimization command is running, `true` estimates congestion as if the rails are present; `false` prevents estimation of rails during congestion analysis. (PKS specific)

Default = `true`

extra_space_for_opt

```
extra_space_for_opt value
```

This is the amount by which optimization may overfill a bin. It is expressed as a percentage (`int`) of acceptable over-utilization of the placement bins. After fixing the violations, placement legalization will move instances out of bins to get rid of overlaps. (PKS specific)

Default: `0`

failsafe

```
failsafe {true | false}
```

In case of a system crash, `ac_shell` has the ability to save the synthesis database generated up to that point. If `failsafe` mode is not set to true, the database must be regenerated by running the commands of the previous session again.

Default: `false`

fanout_load_limit

`fanout_load_limit` *`float`*

This variable has been replaced with the variable `max_fanout_load_limit`.

favor_feedback_cells

`favor_feedback_cells {true | false}`

If set to `true`, will always prefer mapping to registers with enable input, regardless of cost, instead of mapping to registers without enable, resulting in the output of the register feeding back to a MUX in front of the data input of the register. Mapping to such registers with enable inputs, may adversely impact timing.

Default: `false`

fix_multiport_nets

`fix_multiport_nets {true | false}`

If set to `true`, the `do_xform_fix_multiport_nets` command is run automatically as part of `do_optimize`. In this case, any output signal driving multiple output ports is split into different nets by inserting buffers in the paths.

Default: `false`

hierarchy_divider

`hierarchy_divider` *`character`*

When the design hierarchy is dissolved, the convention for naming new design objects can be set to any single ASCII character using the `hierarchy_divider` variable. It is used as a separator in composing a new name from the hierarchical name. In Tcl, the separator needs to be a single character so that it can be represented multiple ways. For example, *c* is the hierarchy divider character you can have `"c"`, `c`, or `{c}`.

Default: _ (underbar)

ignore_enable_logic_for_const_ff_removal

```
ignore_enable_logic_for_const_ff_removal { true | false }
```

When set to `true`, the tool ignores the logic connected to the enable of the flip flop if the flip flop has a constant data input. The constant is then propagated to the fanout of the flip flop, and the flip flop is removed.

Default: `true`

ignore_min_design_rule_violations

```
ignore_min_design_rule_violations { true | false }
```

If set to `true`, the tool ignores minimum slew and minimum capacitance design rules during design rule fixing and optimization.

Default: `false`

ignore_net_area_cost

```
ignore_net_area_cost { true | false }
```

When set to `true`, `net_area_cost` is not included in cost functioning during the optimization.

Default: `false`

instance_generator

```
instance_generator string
```

Choose a scheme for naming automatically generated instances. Use of %d indicates sequentially incrementing counter, starting with 0.

Default: `i_%d`

ipl_pin_limit

`ipl_pin_limit` *`integer`*

For nets with a fanout less than this global, the timing driven incremental placement models each fanout explicitly (PKS specific).

For larger nets, it models only the worst slack pin while calculating instance locations.

Default: `10`

large_fanout_size

`large_fanout_size` *`size`*

Determines which nets go through large-fanout fixing after placement. The maximum allowable value of *`large_fanout_size`* is 200 (the default).

If you are using BuildGates Synthesis or BuildGates Extreme Synthesis, this global has no affect on how large-fanout nets are treated.

Large-fanout nets are handled differently in different parts of the flow:

- Before placement:

  If you are using Cadence PKS, nets with fanout numbers greater than or equal to 200 are marked ideal during pre-placement optimization, and no DRV fixing is done on those nets. The `do_place` command also considers any net with a fanout number greater than or equal to 200 to be ideal. Timing on ideal nets is ignored during placement.

- After placement:

  After placement, the ideal attribute is removed from any large-fanout nets that were marked as ideal during pre-placement optimization. Those nets, as well as nets with a fanout number between 200 and `large_fanout_size` (inclusive), are buffered with a fast buffer tree algorithm at the beginning of post-placement optimization.

  Nets that you marked as ideal remain ideal and are not buffered.

- After large-fanout fixing:

  If the algorithm finishes working on a net, and the net's fanout number is still greater than or equal to 200, it is marked ideal again, and ignored for subsequent optimizations and timing reports. If a net's fanout number is less than `200` after large-fanout fixing, it is no longer treated as a special case by the optimizer.

Default: `200`

line_length

`line_length` *`integer`*

The line length for the messages and reports generated by `ac_shell`. This variable does not control the length of lines for netlist generation. See the `hdl_verilog_out_columns` global.This global does not control report table widths. See the `set_table_style` command.

Default: `80`

logfile

`logfile` *`filename`*

Sets the name of the log file. If *`filename`* can be opened for writing, then the current log file is closed and *`filename`* used. Otherwise the current file is used.

Default: `ac_shell.log`

make_routable_max_over_congestion

`make_routable_max_over_congestion` *`percentage`*

While adjusting the placement for routability, this is the maximum amount by which any bin may be routing congested and the placement still be considered acceptable. Expressed as a percentage. (A value of 10 means that up to 110% of a bin's routing resources may be consumed.) PKS specific.

Default: `15%`

make_routable_max_size

`make_routable_max_size` *`percentage`*

While adjusting the placement for routability, this is the maximum amount by which any bin may be over-utilized and the placement still be considered acceptable. Expressed as a percentage. (A value of 10 means that a bin can be 110% utilized.) PKS specific.

Default: `10%`

make_routable_over_congestion_rate

`make_routable_over_congestion_rate` *`percentage`*

While adjusting the placement for routability, this is the maximum number of bins that may be congested the placement still be considered acceptable. Expressed as a percentage of the total design area. PKS specific.

Default: `10%`

make_routable_oversize_rate

`make_routable_oversize_rate` *`percentage`*

While adjusting the placement for routability, this is the maximum number of bins that may be over-utilized the placement still be considered acceptable. Expressed as a percentage of the total design area (PKS specific).

Default: 6%

manufacturing_grid

`manufacturing_grid` *`value`*

Specifies a manufacturing grid value, in microns. You can use this global to specify a grid value if one does not exist, or to override a library-specified manufacturing grid.

If you use this global, set it before using the `read_lef` command, otherwise the value is ignored.

map_inversion_through_registers

`map_inversion_through_registers { true | false }`

Choose whether `ac_shell` is allowed to store complementary value in the register to perform inversions instead of connecting an inverter to the output. If set to `true`, and `ac_shell` determines that the use of complement value of the register improves area, the inverter is eliminated and the complement value is stored in the register.

Default: `false`

map_to_multibit_registers

`map_to_multibit_registers { true | false }`

If set to true, will attempt to map sets of registers driving a common bus to multibit-register cells in the library, if doing so will give better area. Mapping to multi-bit registers may sometimes impact timing negatively.

Default: `false`

max_capacitance_limit

`max_capacitance_limit` *`float`*

Specifies the maximum capacitance limit on the design, including top level ports.

Default: no limit — infinite

max_fanout_load_limit

`max_fanout_load_limit` *`float`*

Specifies the default fanout load limit used for design rule fixing. The fanout load unit can be a function of the default fanout load unit specified in the library (usually 1), or it can be specified for each input pin.

This global constrains the default load limit for the design. However, if particular technology cells have stricter limits, the limits from the technology library will be honored instead. For instance, if you set `max_fanout_load_limit` to 4 and an IV cell has a `max_fanout` of 2, then 2 will be used for all IV cells in the design.

Default: no limit—infinite

Note: The limits do *not* automatically apply to output ports. You must use the `set_fanout_load` command to set the load on ports.

max_points_to_report

`max_points_to_report` *`integer`*

Specify how many points to include in *`CriticalBegin`* `and` *`CriticalEnd point`* messages during optimization. Default number is 1 but there can be many points included.

Default: 1

max_size_for_std_cells

`max_size_for_std_cells` *int*

Allows a standard cell instance larger than a certain size to be treated as a macro in the PKS database. Set this global immediately after reading in the physical library that contains the particular cell you want treated as a macro.

When this global is set to `0`, it has no effect. When it is set to a positive integer, any standard cell where the cell width and cell height satisfy the following condition will then be marked as a macro:

```
(cell_width > max_size_for_std_cells * max(core_site_width, core_site_height) &&
(cell_height > max_size_for_std_cells * core_site_height)
```

Default: `0`

message_verbosity_level

`message_verbosity_level { [3-9] }`

Setting a high verbosity level implies that the tool gives out more messages. Hence, a message whose verbosity is less than or equal to the verbosity level of the tool will be issued and those who have higher verbosity will be suppressed. So, when specifying the verbosity in the message file, use a HIGH verbosity number to imply messages which are not to be used as often, and a LOW verbosity number for messages which are more important and should be issued. Messages with verbosity set to 0, 1, 2 will always be issued and messages with verbosity set to 9 will hardly ever be issued.

The following guidelines should be used to define the verbosity of a given message in the message file:

These messages when set to `0`, are always issues. So this should be used for internal warning messages.

Default verbosity level for all messages are generated by `ac_shell`.

Default: 7

min_capacitance_limit

`min_capacitance_limit` *`float`*

Specify minimum capacitance limit on all cells.

Default: none

min_porosity_for_over_block_routing

`min_porosity_for_over_block_routing` *`float`*

Sets the minimum porosity a block must have to enable the Steiner router to route over blocks when performing routing estimation.

Refer to the `set_min_porosity_for_over_block_routing` command for more information.

Default: none

multi_segment_default_lut

`multi_segment_default_lut {0 | 1}`

Supports default LUTs in the 5.0 release and backwards compatibility in the 4.0 release.

The 5.0 release introduces a new set of initial default layer usages tables (LUTS), which are hardcoded. At any time you can define your own LUTs and set them as defaults.

LUTS are used for wire resistance and capacitance calculation by the steiner router and by wroute. The main difference of initial default LUTs used in the 5.0 release from ones used in an earlier release, is that the 5.0 release supports different values of layer usages for different lengths of wire segments. As a result, more technology information is taken into account when calculating resistance and capacitance values, so calculated results are more precise.

Use `0` if you are using 4.0 LUTs in the 5.0 release. Use `1` to define LUTs in the 5.0 release.

Default: 1

multiport_fix_buffer_const_nets

```
multiport_fix_buffer_const_nets {true | false}
```

When set to true buffers the generic constant cells feeding the output ports directly. Buffering will eliminate the assign statement in the Verilog output.

naming_style

```
naming_style {vhdl | verilog | none}
```

This variable determines that the I/O of the object names will take place in either VHDL, Verilog, or no naming style. In effect, it reads and prints object names in the specified naming style. The difference in the three options is the way in which the escaping of the illegal string takes place.

For example, if a user runs a `Find` on a net with an illegal name, say a%b (with user ID 12345), the following results will appear:

- Verilog Naming Style

  ```
  find -net 12345
  {\a%b\ }
  ```

- VHDL Naming Style

  ```
  find -net 12345
  {\a%b\ }
  ```

- NONE Naming Style

  ```
  find -net 12345
  {a%b}
  ```

Note the difference in escaping.

Default: `verilog`

net_generator

```
net_generator string
```

Choose a scheme for naming automatically generated internal nets. Use of %d indicates sequentially incrementing counter, starting with 0.

Default: n_%d

no_buffer_at_integration_level

```
no_buffer_at_integration_level { true | false }
```

If set to `true`, optimization will not insert buffers at levels of hierarchy that do not contain any primitive cell.

Default: `false`

no_of_groute_passes_for_cong

```
no_of_groute_passes_for_cong integer
```

Sets the number of passes of optimization that the wroute global router performs when using groute based congestion analysis (PKS specific). Value may be any integer greater than 0.

Default: 2

noise_treat_unconstrained_as_constant

```
noise_treat_unconstrained_as_constant { true | false }
```

If set to `true`, BuildGates Synthesis writes out unconstrained nets as part of the constant TWF net construct. Celtic treats such nets as having zero timing windows.

If set to `false`, BuildGates Synthesis writes out unconstrained nets as having infinite timing windows. Celtic treats such nets as worst-case aggressor and applies rise or fall time and driver impedance accordingly.

This global is used when BuildGates Synthesis writes out the timing windows file that Celtic will use for crosstalk analysis.

Default: `false`

obstruct_pads_completely

```
obstruct_pads_completely {1 | 0}
```

If set to `1`, obstructs the whole area of each IO cell. If set to 0, obstructs the area of each IO cell based on OBS constructs for this cell in the LEF file. This global is to be defined prior to the `read_lef` command, otherwise it will be ignored.

Default: `0`

opt_no_new_instances_at_top_level

```
opt_no_new_instances_at_top_level {true | false}
```

If set to `true`, `optimization` will not insert any new instance at the top most level in the design hierarchy.

`Default`: `false`

opt_no_new_ports

```
opt_no_new_ports {true | false}
```

Instructs the optimization engine to not add any new ports.

Note: Setting this global to `true` may cause some degradation in the quality of results.

Default: `false`

placement_initialize_auto_pass

```
placement_initialize_auto_pass {true | false}
```

This variable is used when the placement setup step in optimization does not meet the incoming acceptance levels. If `true`, the acceptance criteria during the placement setup step are set to the current values. If `false`, the placement setup step will fail and optimization will stop. (PKS specific)

Default: `true`

place_over_utilized

```
place_over_utilized {true | false}
```

If set to `true`, placement will continue even though the utilization is over 100%. Placement quality will be compromised and the resultant placement will not be overlap-free. (PKS specific).

Default: `false`

preserve_constant_flops

```
preserve_constant_flops {true | false}
```

If set to `false`, the do_xform_propagate_constants command will remove constant flip-flops in additions to constant latches during optimization.

Default: `true`

remove_filler_cells_for_opt

```
remove_filler_cells_for_opt { true | false }
```

When set to `false`, prevents filler cells from being removed. When set to `true` (the default), filler cells are removed during optimization.

Default: `true`

repair_file_may_change_clock_nets

```
repair_file_may_change_clock_nets {on | off}
```

By default, the do_xform_run_repair_file command does not change clock nets. If you want crosstalk violations to be fixed for clock nets, set this global to `on`.

Default: `off`

reset_lfo_ideal_nets_after_opt

```
reset_lfo_ideal_nets_after_opt {on | off}
```

By default, large-fanout nets that cannot be fixed are marked as ideal. When this global is set to `on`, the ideal attribute is removed from such nets after optimization. If you then run a timing report, these nets will probably be reported as having very large delays.

See also large_fanout_size on page 39 and set_ideal_net_after_lfo_fixing_fails on page 49.

Default: `off`

resize_dont_modify_instances

```
resize_dont_modify_instances  {true | false}
```

If this global is set to `true`, it allows even instances with the `dont_modify` property to be resized during optimization. When this variable is set to `false` , instances which are `dont_modify`, are not allowed to be resized.

Default: `false`

resize_registers_in_clock_propagated_mode

```
resize_registers_in_clock_propagated_mode {true | false}
```

When set to `false`, disables resizing of registers when the clock network is in propagated mode. This can be useful when maintaining the exact capacitive load of the clock tree is a concern.

Default: `true`

set_ideal_net_after_lfo_fixing_fails

```
set_ideal_net_after_lfo_fixing_fails {on | off}
```

By default, large-fanout nets that cannot be fixed by the fast buffer tree algorithm after placement are marked as ideal. When this global is set to `off`, these nets are not marked as ideal, and therefore they undergo DRV fixing during optimization. This will probably result in more large-fanout nets being fixed at the cost of longer runtime.

See also large_fanout_size on page 39 and reset_lfo_ideal_nets_after_opt on page 48.

Default: on

smoothen_area_gap

```
smoothen_area_gap percentage
```

While adjusting the placement for optimization, this is the amount of empty row area that PKS creates in each bin. Expressed as a percentage of the bin row area. (PKS specific)

Default: 1%

smoothen_utilization_only

```
smoothen_utilization_only { true | false }
```

If `true`, adjust placement during optimization to eliminate any local utilization problems. If `false`, adjust the placement to address both utilization and congestion problems. (PKS specific)

Default: `true`

steiner_pessimism_length_limit

```
steiner_pessimism_length_limit value
```

Note: Using this variable is not recommended. It should be limited to debug use only and must be used in conjunction with the steiner_pessimism_scale_factor global.

Provides a mechanism for adding additional pessimism to long nets during the pre-route (Steiner) stage. Adding additional pessimism to long nets helps to reduce any inconsistencies between the Steiner route and global route values that may cause routes to detour or generate different route patterns. During Steiner analysis, RC values are extracted for each wire segment based on unit wire resistance (r) and capacitance per unit length (c).

Setting this global forces the tool to increase the r and c unit values, which provides the additional pessimism for long nets. The r and c unit values are calculated for any net whose total wire length (w) exceeds the steiner_pessimism_length_limit (L).

```
r = r + (w/L)*f*r
c = c + (w/L)*f*c
```

where f = steiner_pessimism_scale_factor.

steiner_pessimism_scale_factor

```
steiner_pessimism_scale_factor value
```

Note: Cadence does not recommend the use of this global. Is should be limited to debug use only and must be used in conjunction with the steiner_pessimism_length_limit global.

Provides the scaling factor used with the steiner_pessimism_length_limit global.

steiner_stitch_partial_routes

```
steiner_stitch_partial_routes {true | false}
```

Instructs the steiner routing engine to retain any pre-routes or partial routes defined in the database and to stitch the missing routes with steiner routes.

Default: true

time_budget_min_size

`time_budget_min_size` *`integer`*

Controls the hierarchical scope at which the optimization will be performed within the `do_optimize -time_budget` compile. The value of the variable is the minimum number of cell instances which must be present at or beneath the module to be compiled. Increasing the value of this variable will force the time budgeting and optimization to be run only at higher levels of the design hierarchy.

Default: `100`

time_budget_stop_before_uniquification

`time_budget_stop_before_uniquification { true | false }`

Exits the `do_optimize -time_budget` compile loop prior to the uniquification and top-level hierarchical compile phase when set to true, which is useful if you wish to experiment with multiple passes of `do_optimize -time_budget`.

Default: `false`

topt_assert_default_design_rules_on_ports

`topt_assert_default_design_rules_on_ports { true | false }`

If this global is set to `true` and there are top-level ports that do not have design rules, the software automatically asserts the appropriate design rule, based on the loosest design rule constraints from the family of usable buffers or inverters.

Default: `true`

topt_macro_cell_outputs_are_tristates

`topt_macro_cell_outputs_are_tristates { on | off }`

BG/PKS disallows the buffering of nets driven by tristate drivers and driving top-level output ports, conservatively assuming that these nets could be joined by other tristate sources outside the current design being synthesized (this assumption could be disabled by turning off another global: `topt_no_external_sources_at_outputs`).

BG/PKS attempts to infer whether a driver is tristate or not from the functional description of the cell in the library or from the existence of a tristate-trigger arc in the timing description of the cell in the library. When neither of these is present, BG/PKS assumes the cell is not a tristate driver. Functional descriptions are usually absent for macro cells. Turning on this global will instruct the tool to assume that all cells without functional descriptions are tristate drivers.

Default: `off`

topt_no_external_sources_at_outputs

```
topt_no_external_sources_at_outputs { true | false }
```

Enables buffering of tristate nets that drive primary output ports when set to `true`.

Default: `false`

Note: When you can not remove a combinational cell type (non-tristate nets), when the max transition violation on a net is connected directly to an output port, even by using the `do_xform_fix_design_rule_violation` command, set the global `topt_no_external_sources_at_outputs` to `true` to tell the tool that there are no other external sources driving the output, so it is safe to place a buffer on the net.

unmap_generic_flip_flop

```
unmap_generic_flip_flop {true | false}
```

When set to `true`, unmaps generic macro flip-flops before structuring. This is useful when an input signal goes to both data and synchronous enable logic, and converges then to the data pin of a flip-flop. Without unmapping the macro flip-flop, structuring would not see the complete data logic cone during logic optimization.

Default: `false`

use_drive_cell_design_rules

```
use_drive_cell_design_rules {true | false}
```

If set to `true`, design rules, such as `fanout_load_limit`, `slew_limit` and `max_capacitance_limit` are picked up from the drive cells as well. By default, design rules are not picked up from the drive cells.

This global refers to input ports. Setting this global depends on where an assertion comes from and how fixed the assertion is. For example, if the current design is a sub block and the

outside world is optimized, take the DRVs from the asserted drive cell into account. However, if for example, the outside world is not yet optimized and the drive cell might be low drive strength, or the final driver is not the asserted cell, but a pad cell from another library, then ignore the implied DRV violations.

Default: `false`

use_groute_based_cong_analysis

```
use_groute_based_cong_analysis {0 | 1}
```

When set to `1`, enables a global route based congestion analysis. When set to `0`, enables a steiner based congestion analysis. (PKS specific.)

Default: `0`

use_lef_area

```
use_lef_area {true | false}
```

The reporting of area is strictly controlled by this global. When set to `false`, you force the tool to generate area cost functions using the logical library area rather than the default physical library area.

Default: `true`

warn_about_lfo_ideal_nets_after_opt

```
warn_about_lfo_ideal_nets_after_opt {on | off}
```

Issues warnings after `do_optimize` (post-placement) if any large fanout nets (nets with fanout greater than or equal to 200) are still marked `ideal`.

Default: `on`

wired_logic_resolution

```
wired_logic_resolution { and | or }
```

Choose a default resolution function for nets driven by multiple drivers.

Default: `and`

write_gcf_default_version

`write_gcf_default_version`

Sets the default version for writing gcf.

Default: 1.4

3

Ambitware Globals

aware_adder_architecture

```
aware_adder_architecture {ripple | csum | csel | cla | fcla}
```

Sets the default adder architecture for (final) adders. The `fcla` option is only available with the datapath option.

Default: `cla` for AWCL and `fcla` for AWDP

aware_carrysave_inferencing

```
aware_carrysave_inferencing {true | false}
```

Turns on operator merging in multi-fanout scenarios when set to `true.` Turns off operator merging in multi-fanout scenarios when set to `false`.

Default: `true`

aware_dissolve_width

```
aware_dissolve_width positive_integer
```

Sets the limit for the width of the operator units (such as adders, subtractors, incrementors, decrementors, and comparators) to be dissolved. The value is an integer indicating the number of bits for the size of a ripple adder. All adders with the gate count less than or equal to the ripple adder of the specified bit width will be dissolved.

Default: 4

Note: The dissolve width is based on the literal count of the operator and is influenced by the architecture. For example: a 16 bit adder implemented using the cla architecture, is about the same size as a 24-bit ripple adder, therefore the *`integer`* value should be 24.

aware_divider_architecture

```
aware_divider_architecture {radix_2 | radix_4}
```

Specifies the divider architecture to be implemented by datapath synthesis. The `radix_4` architecture is faster but also consumes more area than that of `radix_2`. The number of stages employed by the `radix_2` architecture equals the number of bits in the dividend. The number of stages employed by the `radix_4` architecture equals half the number of bits in the dividend.

Default: `radix_4`

aware_implementation_selection

```
aware_implementation_selection {true | false}
```

When set to true, the implementation of datapath components are automatically selected based on the constraints. If set to false, the detailed implementation of each component may be insensitive to the timing characteristics of a component's surrounding control logic. This command is only available with the datapath option.

Default: `true`

aware_library_search_order

```
aware_library_search_order logical_names
```

Sets the order in which aware libraries are searched for components. If two libraries contain a component with the same name, the one found first will be used. This command is only available with the datapath option.

Default: `AWARITH AWLOGIC`

aware_merge_operators

```
aware_merge_operators { true | false }
```

Controls operator merging. When `ac_shell` is invoked with the datapath option, operator merging is enabled by default. You can disable operator merging with the command `set_global aware_merge_operators false` before calling `do_build_generic`. You can also control operator merging using a pragma that forces merging to stop at the operator on which the property is attached via the pragma. The following Verilog pragma results in a merged implementation of the expression:

```
assign z = a * //ambit synthesis merge_boundary
b + c;
```

This may be useful in situations where you do not want the tool to merge some particular operator with other downstream operators.

The tool does not limit the scope of operator merging to arithmetic expressions defined by a single HDL statement. Operator merging is done on the CDFG (Control Data Flow Graph) representation derived from the HDL and can span operators in multiple HDL statements.

Default: `true`

aware_multiplier_architecture

```
aware_multiplier_architecture {booth | non_booth | auto}
```

Sets the default multiplier architecture for multipliers. This command is only available with the datapath option.

Default: `auto`

aware_mux_dissolve_size

```
aware_mux_dissolve_size positive_integer
```

Sets the minimum size, in terms of the number of data inputs, for which a multiplexer can be determined to be “map-only” by the AmbitWare multiplexer generator. Multiplexers that have fewer data inputs than the specified `aware_mux_dissolve_size` are dissolved at the beginning of the `do_optimize` command and optimized (structured and mapped) within the context of the surrounding logic.

Default: 8

aware_optimize_merge_boundary

```
aware_optimize_merge_boundary { true | false }
```

During technology independent synthesis, or generic synthesis, datapath operator merging is used to synthesize multiple arithmetic operators in the RTL design in order to eliminate the delays associated with the generation of intermediate results. Merging must be performed safely to ensure functional equivalence between the RTL design and the synthesized netlist. Therefore, during synthesis merging is performed only on those operators that preserve this functional equivalence.

When set to `true`, BuildGates attempts to identify operations that represent false merging bottlenecks to enhance the scope of operator merging during generic synthesis. The types of operations that can represent false merging bottlenecks include some situations of bit-selects, sign and width extensions, concatenation, shifts by constants, and arithmetic rounding.

The following example illustrates an instance of false merging bottlenecks in an RTL description and how the `aware_optimize_merge_boundary` global can enhance operator merging by recognizing these bottlenecks:

```
module test(a,b,c,res);
     input [5:0] a,b,c,d;
     output [6:0] res;

     wire [7:0] temp1 = {a,1'b0}+ (b << 1);
          assign res = (c << 3) + temp1[6:0] + d;
endmodule
```

The bit-select in `temp1[6:0]`, the constant left shifts in `(b << 1)`and `(c << 3)`, the concatenation in `{a,1'b0}`, and the truncation in the assignment to `res`, are not bottlenecks to merging. When these false bottlenecks are removed, the synthesis of the above example results in only one adder module that adds all the four addends.

The `aware_optimize_merge_boundary` global can help in situations in which the RTL is generated automatically by another EDA tool (for example, a system level design tool that generates a HDL description of the design) or when the RTL design style incorporates explicit operations, like concatenations and bit-selects, to do sign and width extensions.

In some situations, keeping this global to `false` can lead to worse quality of results than if it were set to `true` depending on the extent in which false merging bottlenecks appear in the given RTL.

Default value: `false`

aware_suffix_module_name

```
aware_suffix_module_name { true | false }
```

During generic synthesis, BuildGates creates internal module hierarchies in the design in order to facilitate certain optimizations. These hierarchies appear as module names in the generic netlist or in a netlist obtained during the early stages of optimization. If this global is set to `false`, the names of the created datapath and mux related computations only receive a numerical suffix. For example:

```
AWDP_partition_0, AWDP_partition_1,...,AWMUX_2_10
```

It may be difficult to correspond these generated names to the original RTL. When set to `true`, this global improves the transparency of the generated names with the original RTL by appending the name of the enclosing module to every generated datapath and mux hierarchy name. The following example illustrates a generated module name, with the global set to `false`, for a computation inside the user module named "foo":

```
AWDP_partition_0
```

If the global had been set to `true`, the above name would look like:

```
AWDP_partition_0_foo
```

Default value: `false`

4

CTPKS Globals

ct_inverted_polarity_suffix

`ct_inverted_polarity_suffix` *`string`*

When set, indicates the string suffix to be added to a connector name when its polarity needs to be changed due to the insertion or deletion of inverters.

Default: `-inv`

ctpks_default_global_clock_model_file

`ctpks_default_global_clock_model_file` *`filename`*

When this variable is set and pointing to an existing file, the `do_build_clock_tree` command (without the `-clock_model switch`) loads the general clock model definitions from the specified file.

Default: none

ctpks_ignore_dont_utilize_property

`ctpks_ignore_dont_utilize_property {false|true}`

When `true`, allows CTPKS to use buffers regardless of the `dont_utilize` property, if they either have no `ct_dont_utilize` attribute set or have the `ct_dont_utilize` attribute set to `false`.

Default: `false`

ctpks_ignore_false_path

`ctpks_ignore_false_path {true | false}`

When true, tells the `report_clock_tree` and `do_build_clock_tree` commands to recognize the clock tree and calculate its timing despite timing exceptions such as false paths.

Default: `false`

ctpks_ignore_max_fanout

```
ctpks_ignore_max_fanout {true | false}
```

When set to `true`, CTPKS ignores the `max_fanout` rule when building the clock tree.

Default: `true`

ctpks_port_generator

```
ctpks_port_generator string
```

Specifies a naming convention for ports created by the `do_build_clock_tree` and `do_build_physical_tree` commands. Use of %d indicates sequentially incrementing counter, starting with 0.

When you specify this global,

- All new ports follow the port naming convention defined with `ctpks_port_generator` (for example, `CTPKS_2`).
- All new inverted ports follow the naming conventions defined in `ctpks_port_generator` and `ct_inverted_polarity_suffix` (for example, `CTPKS_2_inv`).
- All original ports keep their names (for example, `clk`).
- All original inverted ports keep their names (for example, `clk_inv`).
- Associated nets follow previous naming conventions.

Default: Ports are created using PKS naming convention (p_%d).

ctpks_rename_net_to_port_nets

```
ctpks_rename_net_to_port_nets {on | off}
```

Prevents CTPKS from creating a new net with same name as an existing hierarchical port to which the net is not connected.

In such a situation, there is an naming inconsistency in the `write_def` and `write_verilog` commands. When `write_verilog` processes such a net, it will rename it `net_name__ac__`, because in Verilog, a net which has the same name as a port is implicitly connected to it. The `write_def` command retains the name of the net without

changing it. This creates a mismatch. Setting the `ctpks_rename_net_to_port_nets` global to `on` eliminates this inconsistency.

Default: `off`

ctpks_repeaters_number_limit

`ctpks_repeaters_number_limit` *`value`*

When set, indicates the maximum number of buffers or inverters to be inserted to buffer a single net. When this value is exceeded during trials, the chain of buffers is removed.

Default: 20.

ctpks_write_def_source_timing

`ctpks_write_def_source_timing {true | false}`

When set to true, writes " + SOURCE TIMING" in DEF on nets and components.

The default behavior of the CTPKS commands (`do_build_clock_tree` and `do_build_physical_tree`) are *not* to attach "+SOURCE TIMING" in DEF on nets and components. When set to `true`, these CTPKS commands attach this property to created nets and components. The property is written in a DEF file when the command `write_def` is invoked.

Default: `false`

5

Distributed Synthesis Globals

dist_batch_queue

`dist_batch_queue` *name*

Sets the name of the LSF batch queue to be used when `dist_launch_mode` is set to `batch`. If the value is not null, then batch queue job handling is enabled. There is no default value.

dist_bits

`dist_bits` *p2*

Specifies the default number of bits for all `ac_shell` sessions, where *p2* is a power of 2. The default setting can be overridden on a specific session using the `set_dist_bits` command.

Default: `32` for 32-bit `ac_shell` and `64` for 64-bit `ac_shell`

dist_capture_job_histogram

`dist_capture_job_histogram {true | false}`

When set to `true`, distributed synthesis generates graph data in the GUI of the actual job distribution.

Default: `false`

dist_default

`dist_default { on | user | off }`

Controls whether a command runs in distributed mode.

- `on`

  Command runs in distributed mode, even if the `-distributed` option is not specified.

- `user`

  Command runs in distributed mode only if `the -distributed` option is specified.

- `off`

  No command runs in distributed mode, even if the `-distributed` option is specified.

Default: `user`

dist_embargo_delay

```
dist_embargo_delay time
```

Prevents jobs from launching on a host for a specific period of time when another job has just been launched. This allows the host load to catch up.

Default: `0:30` (30 sec.)

dist_enable_final_top_down

```
dist_enable_final_top_down { true | false }
```

If set to `true`, a final top-down pass of timing optimization is performed. This top-down optimization is performed using the `do_xform_timing_correction` command This can improve slack for designs with difficulty in meeting timing constraints.

Default: `false`

dist_granularity

```
dist_granularity { medium | fine | coarse }
```

Sets the distributed flow granularity. For very large designs, use the `coarse` value. For smaller designs, `fine` is recommended.

Default: `medium`

dist_kill_signal

```
dist_kill_signal signal
```

Sets the signal used to kill a job. *`signal`* is the name or number of a (UNIX/POSIX) signal used to kill all local and remote job processes. Signals `SIGHUP`, `SIGINT`, `SIGQUIT`, or `SIGTERM` are recommended.

Default: `SIGTERM`

dist_kill_verbose

```
dist_kill_verbose { true | false }
```

Determines what information distributed synthesis displays when it kills a job on a remote host. When set to `true`, distributed synthesis displays the names of the machines on which jobs are currently running, the pid, and host name information. When set to `false`, distributed synthesis does not display this information.

Default: `false`

dist_launch_delay

```
dist_launch_delay time
```

Sets the maximum delay time between checking for ready jobs. The master `ac_shell` waits for a this amount of time before launching more jobs. At intervals, it also reports a list of currently running jobs, also see `dist_summary_delay`.

Default: `5:00` (`5 min`.)

dist_launch_mode

```
dist_launch_mode { batch | host_list }
```

Sets the launch mode for remote jobs to `batch` or `host_list`. Batch mode uses LSF to run remote jobs, whereas host list mode uses built-in load balancing.

Default: `host_list`

dist_launch_timeout

```
dist_launch_timeout time
```

Sets the launch idle time-out period. If the master `ac_shell` cannot launch any remote jobs within this time period (since the last job was launched), the distributed command stops.

Default: `30:00` (`30 min`.)

dist_max_failures

`dist_max_failures` *integer*

Specifies the limit for the number of recoverable errors allowed prior to or during a job run. When the failure limit is met, the job is rerun, provided the number of retries has not yet been exceeded.

Default: `2`

dist_max_jobs

`dist_max_jobs` *integer*

Specifies the maximum number of jobs that can run at one time by a remote `ac_shell`. At least one job must be running.

Default: `4`

dist_max_load

`dist_max_load` *percent*

Jobs are not launched on hosts with a CPU utilization that exceeds this value.

Note: The load on multi-CPU hosts is normalized to a single CPU. If the load of a two CPU host is 50%, that means that one CPU is busy and the other is idle. The normalized load in this case is 0% to indicate that one CPU is idle.

Default: 15%

dist_max_restarts

`dist_max_restarts` *n*

Limits the number of times jobs can be restarted. A value of zero indicates the number of restarts is unlimited. If the number of restarts is exceeded, the job will not run and an error is returned.

Default: `0`

dist_min_cpus

`dist_min_cpus` *`integer`*

Specifies the minimum number of CPUs that must be accessible to run distributed synthesis. If the minimum requirement is not met, the `-distributed` option is ignored.

Default: 2

dist_min_jobs

`dist_min_jobs` *`integer`*

Specifies the minimum number of jobs that can run in distributed mode. If the width of the design hierarchy is less than this value, the optimization command is not run in distributed mode. If set to less than 2, the tool issues a warning and prevents the command from running in distributed mode.

Default: 2

dist_rlimit

`dist_rlimit` *`time`*

Sets a time limit for distributed synthesis commands in remote jobs. For example, if you enter the following command, `set_global dist_rlimit 2:00`, each remote optimization command is limited to no more than two minutes CPU time.

dist_nice

`dist_nice` *`integer`*

Specifies the (UNIX/POSIX) `nice` value used when running remote jobs. This is needed to run remote jobs at a lower priority on a workstation or when running remote jobs on the master host. If you set this variable to `0`, distributed synthesis does not lower the priority of remote jobs.

Default: `0`

dist_remsh_timeout

`dist_remsh_timeout [`*`hours`*`]:[`*`minutes`*`]:`*`seconds`*`.[`*`milliseconds`*`]`

Defines the time-out period for remote shell commands. The master `ac_shell` invokes remote shell commands to execute a command on a remote host, for example, to get the load of a host. Because remote hosts can be busy or off line, remote shell commands are invoked with a time-out option to prevent hanging the master `ac_shell`. The command `get_global dist_remsh_timeout` returns the time-out period.

Default: `2.0` seconds.

On busy networks, a larger value may be appropriate (3 to 4 seconds). If a lot of time out warning messages occurs during a distributed run, you may want to increase the value of `dist_remsh_timeout`.

Note: While a time-out value of zero is allowed, it is not recommended as it may cause the master `ac_shell` to hang or wait indefinitely.

dist_restart_delay

`dist_restart_delay` *`hh:mm:ss`*

Specify the time period to delay the restart of a job stopped by the `dist_restart_signal` global.

Default: `00:00:00`

dist_restart_embargo

`dist_restart_embargo` *`hh:mm:ss`*

Specify the time period to wait before allowing a host to run another job after a job was killed on it. Effectively, this variable takes a host off-line for the time period specified.

Default: `00:00:00`

dist_restart_signal

`dist_restart_signal {NULL | QUIT | HUP | TERM | USR1 | USR2}`

Allows the master `ac_shell` to kill a job running on a remote machine. The kill request can be input one of three ways: using the distributed synthesis `restart script`, with the `kill` command from the Unix shell, or with the `bkill` command if running in batch mode. For more

information on the `restart script`, refer to *Restart Script* in the *Distributed Processing of BuildGates Synthesis*.

The signals `QUIT, HUP, TERM, USR1,` and `USR2` are “catchable,” which enables the an orderly exit of the remote job. The master `ac_shell` will reschedule the killed jobs either on the same or different host.

Default: `NULL` (no restart allowed)

dist_retries

`dist_retries` *integer*

Specifies the maximum number of times that distributed synthesis can try to run a job after the initial job run failed.

Default: 2

dist_shutoff

`dist_shutoff` *time*

Sets the shutoff time-out period for a remote job. If a remote job does not finish in the time-out period, after the final heartbeat has been received by the master `ac_shell` the remote job is killed. Setting the value to `0.0` turns off this option.

Default: `5:00` (`5 min`.)

dist_startup

`dist_startup` *time*

Sets a startup time-out period for a remote job. A remote job should start to run and send the first heartbeat to the master `ac_shell` within this time-out period. If a job does not start within this period, it is considered to have failed. The startup time-out period starts when the job is launched. Setting the value to `0.0` turns off this option.

Default: `0:00`

dist_std_file

`dist_std_file` *`filename`*

Specifies the name of the remote job log file, which contains the log of the remote `ac_shell` and additional information, such as the LSF log, when batch mode is used. The name of the log file is unique and contains the job identifier, the name of the master host, and the process identifier on the master `ac_shell`.

Default: `.job` *`job_id`* `.` *`mhost`* `.` *`mpid`*`.std`

dist_stop_after_mapping

`dist_stop_after_mapping { true | false }`

When set to `true`, optimization stops after the mapping phase, and no timing optimization is done.

Default: `false`

dist_summary_delay

`dist_summary_delay` *`time`*

Sets the time interval between printing summary reports on currently running jobs by the master `ac_shell`.

Default: `0:30` (30 sec.)

dist_timeout

`dist_timeout` *`time`*

Sets a time-out for the run time of a remote job. If the run time of a job exceeds the time-out period, the master `ac_shell` kills the job and reruns it on another host. The run time of a job starts when the master `ac_shell` receives the first heartbeat. Setting the value to `0.0` turns off this option.

Default: `0:00`

dist_uniquify

```
dist_uniquify { early | late | none }
```

Controls the point at which uniquification occurs during optimization. When the value is set to `early`, uniquification occurs before structuring. When set to `late`, it occurs after mapping. Otherwise, no uniquification occurs.

Default: `early`

dist_verbose

```
dist_verbose integer
```

Determines the level of detail for job messages in the master `ac_shell` log file. A minimum number of job messages are provided when the default value is used. Zero means no reports.

Default: `1`

dist_weight

```
dist_weight nonnegative_integer
```

Sets the relative size of the job, which can determine the hosts on which the job can run. A job can run on a host only if the weight of the host is the same or greater than the weight of the job. Zero indicates that the job can run on any host.

Default: 0

6

HDL Globals

edifin_bus_dimension_separator_style

`edifin_bus_dimension_separator_style` *`string`*

Specifies the two characters used to denote an index or range specification for accessing elements of a bus in EDIF designs being read in.

The text between the two characters is interpreted as an index to an element of the bus, for example, `X[2]` or a range specification, for example, `X[2:4]`.

Default: `[]`

edifin_bus_range_separator_style

`edifin_bus_range_separator_style` *`string`*

Characters used to separate names of bussed ports and nets in EDIF designs being read in.

Default: `:` (colon)

edifin_ground_instance_name

`edifin_ground_instance_name` *`string`*

Name of the instance that represents ground when `edifin_power_and_ground_representation` is set to `instance`.

Default: `GND`

edifin_ground_net_name

`edifin_ground_net_name` *`string`*

Name of the ground net when `edifin_power_and_ground_representation` is set to `net`.

Default: `""`

edifin_ground_net_property_name

`edifin_ground_net_property_name` *`string`*

Name of the property that nets must have to be interpreted as ground when `edifin_power_and_ground_representation` is set to `net`. Used in conjunction with `edifin_ground_net_property_value`.

Default: `default`

edifin_ground_net_property_value

`edifin_ground_net_property_value` *`string`*

Value of the property that nets must have to be interpreted as ground when `edifin_power_and_ground_representation` is set to `net`. Used in conjunction with `edifin_ground_net_property_name`.

Default: `logic_0`

edifin_ground_pin_name

`edifin_ground_pin_name` *`string`*

Name of pin of the instance that represents ground when `edifin_power_and_ground_representation` is set to `instance`.

Default: `GND`

edifin_ground_port_name

`edifin_ground_port_name` *`string`*

Name of the ground port when `edifin_power_and_ground_representation` is set to port.

Default: `GND`

edifin_power_and_ground_representation

`edifin_power_and_ground_representation {net | port | instance |none}`

Representation of power and ground in EDIF designs being read in. Allowable values are `none, net, port,` and `instance.`

`Default`: `net`

edifin_power_instance_name

`edifin_power_instance_name` *`string`*

Name of the instance that represents power when `edifin_power_and_ground_representation` is set to `instance`.

Default: `PWR`

edifin_power_net_name

`edifin_power_net_name` *`string`*

Name of the power net when `edifin_power_and_ground_representation` is set to `net`.

Default: `""`

edifin_power_net_property_name

`edifin_power_net_property_name` *`string`*

Name of the property that nets must have to be interpreted as power when `edifin_power_and_ground_representation` is set to `net`. Used in conjunction with `edifin_power_net_property_value`.

Default: `default`

edifin_power_net_property_value

`edifin_power_net_property_value` *`string`*

Value of the property that nets must have to be interpreted as power when `edifin_power_and_ground_representation` is set to `net`. Used in conjunction with `edifin_power_net_property_name`.

Default: `logic_1`

edifin_power_pin_name

`edifin_power_pin_name` *`string`*

Name of pin of the instance that represents power when `edifin_power_and_ground_representation` is set to `instance`.

Default: `PWR`

edifin_power_port_name

`edifin_power_port_name` *`string`*

Name of the power port when `edifin_power_and_ground_representation` is set to port.

Default: `PWR`

edifout_array

`edifout_array {true | false}`

Determines whether the arrays will be represented as single.

Default: `true`

edifout_designs_cell_name

`edifout_designs_cell_name` *`string`*

Name of the `cellRef` used in the EDIF `design` construct.

Default: `""`

edifout_designs_library_name

`edifout_designs_library_name` *`string`*

Name of the `libraryRef` used in the EDIF `design` construct

Default: `""`

edifout_designs_name

`edifout_designs_name` *`string`*

Name used in EDIF `design` construct.

Default: `""`

edifout_ground_cell_name

`edifout_ground_cell_name` *`string`*

Name of the cell, an instance of which represents ground when `edifout_power_and_ground_representation` is set to `instance`.

Default: `GND`

edifout_ground_instance_name

`edifout_ground_instance_name` *`string`*

Name of the instance that represents ground when `edifout_power_and_ground_representation` is set to `instance.`

`Default`: `GND`

edifout_ground_net_name

`edifout_ground_net_name` *`string`*

Name of the ground net when `edifout_power_and_ground_representation` is set to `net`.

Default: `""`

edifout_ground_net_property_name

`edifout_ground_net_property_name` *`string`*

Name of the ground net when `edifout_power_and_ground_representation` is set to `net`.

Default: `""`

edifout_ground_net_property_value

`edifout_ground_net_property_value` *`string`*

Value of the property that nets must have to be interpreted as ground when `edifout_power_and_ground_representation` is set to `net`. Used in conjunction with `edifout_ground_property_name`.

Default: `logic_0`

edifout_ground_pin_name

`edifout_ground_pin_name` *`string`*

Name of pin of the instance that represents ground when `edifout_power_and_ground_representation` is set to instance.

Default: `GND`

edifout_ground_port_name

`edifout_ground_port_name` *`string`*

Name of the ground port when `edifout_power_and_ground_representation` is set to `port`.

Default: `GND`

edifout_power_and_ground_representation

```
edifout_power_and_ground_representation {net | port | instance | none}
```

Representation of power and ground in EDIF designs being written out. Allowable values are `none`, `net`, `port`, and `instance`.

Default: `net`

edifout_power_cell_name

```
edifout_power_cell_name string
```

Name of the cell, an instance of which represents power when `edifout_power_and_ground_representation` is set to `instance`.

Default: `PWR`

edifout_power_instance_name

```
edifout_power_instance_name string
```

Name of the instance that represents power when `edifout_power_and_ground_representation` is set to `instance`.

Default: `PWR`

edifout_power_net_name

```
edifout_power_net_name string
```

Name of the power net when `edifout_power_and_ground_representation` is set to `net`.

Default: `""`

edifout_power_net_property_name

`edifout_power_net_property_name` *`string`*

Name of the property that nets must have to be interpreted as power when `edifout_power_and_ground_representation` is set to `net`. Used in conjunction with `edifout_power_property_value`.

Default: `default`

edifout_power_net_property_value

`edifout_power_net_property_value` *`string`*

Value of the property that nets must have to be interpreted as power when `edifout_power_and_ground_representation` is set to `net`. Used in conjunction with `edifout_power_property_name`.

Default: `logic_1`

edifout_power_pin_name

`edifout_power_pin_name` *`string`*

Name of pin of the instance that represents power when `edifout_power_and_ground_representation` is set to `instance`.

Default: `PWR`

edifout_power_port_name

`edifout_power_port_name` *`string`*

Name of the power port when `edifout_power_and_ground_representation` is set to `port`.

Default: `PWR`

edifout_properties

`edifout_properties {true | false}`

Determines whether the properties associated with netlist objects will be written out or not. Properties such as `logic_0` or `logic_1` are written out in the netlist irrespective of the status of this global because they represent the constants `0` and `1` in the design. Removing these properties would make the design functionally incorrect.

Default: `false`

hdl_array_generator

`hdl_array_generator` *`string`*

Chooses a scheme to name individual bits of array ports and registers. The string argument must include `%s` to indicate the record name of the bus signal, and `%d` to indicate the array index. Set this global before performing `do_build_generic`. Related command are `do_blast_busses` and `do_rename`.

Default: `%s_%d`

hdl_common_subexpression_elimination

`hdl_common_subexpression_elimination {true | false}`

Determines whether common sub-expression elimination should be performed on HDL operators before generating the generic netlist. The HDL operators considered include arithmetic operators (for example, `+`, `-`, `*`, `abs`), shift operators (for example, <<, >>), relational operators (for example, >,>, =), and user-defined operators.

Default: `true`

hdl_cse_for_registers

`hdl_cse_for_registers {true | false}`

When set to true and `hdl_common_subexpression_elimination` is set to `true`, registers are also considered for elimination.

Default: `false`

hdl_error_on_latch

```
hdl_error_on_latch { true | false }
```

When set to `true`, an error is issued if a latch is inferred for a design.

Default: `false`

hdl_extract_sum_of_products_logic

```
hdl_extract_sum_of_products_logic {true | false}
```

When set to `true` prior to `do_build_generic`, variables that are assigned only constant values within a case statement are represented by Boolean equations that are in a sum-of-products (SOP) form. SOP logic that is identified and extracted during `do_build_generic` is minimized during `do_optimize` with specialized and efficient logic optimization techniques. For more information, see *"Extraction of Sum-of-Products Logic" in HDL Modeling for BuildGates Synthesis.*

Default: `true`

hdl_ff_auto_sync_set_reset

```
hdl_ff_auto_sync_set_reset { true | false }
```

When set to `true`, the set/reset signal will be preserved with the flip-flop during generic optimization. This prevents its incorporation into random logic during structuring. This global only notes an implementation preference. It does not force the tool to honor the global. Therefore, in some scenarios it could be ignored if such an omission provides a better quality netlist. To force an implementation, use the set_register_type command.

Default: `false`

hdl_keep_feedback

```
hdl_keep_feedback { true | false }
```

When set to `false`, `do_build_generic` eliminates explicit feedback loops (in other words, assignments like `q <= q`) for flip-flops, and creates a synchronous enable instead.

Default: `false`

hdl_latch_auto_async_set_reset

```
hdl_latch_auto_async_set_reset { true | false }
```

When set to `true`, the `do_build_generic` command uses asynchronous set and reset pins, rather than data input pins, to implement all asynchronous set and reset operations for latches. This global only notes an implementation preference. It does not force the tool to honor the global. Therefore, in some scenarios it could be ignored if such an omission provides a better quality netlist. To force an implementation, use the set_register_type command.

Default: `false`

hdl_max_loop_limit

```
hdl_max_loop_limit integer
```

Determines the maximum number of iterations for unfolding a loop construct of any type. If the limit is exceeded an error is issued.

Default: `1000`

hdl_max_recursion_limit

```
hdl_max_recursion_limit integer
```

Sets the maximum number of elaborations for recursive instantiations to prevent possible infinite recursions.

Default: `1000`

hdl_optimize_conditional_computations

```
hdl_optimize_conditional_computations {true | false}
```

When set to `true`, a sequence of conditional computations in the control data flow graph is replaced with an equivalent subgraph that represents an arithmetic operation. This process occurs during generic synthesis. This transformation helps minimize area or improve timing or both.

Default: `false`

hdl_preserve_unused_registers

```
hdl_preserve_unused_registers { true | false }
```

When set to `true`, the software not remove unused registers (latches and flip-flops) that do not, directly or indirectly, affect any outputs. This can be used, for example, to keep registers inserted for the only purpose of observing internal nets through scan chains in test mode.

Default: `false`

hdl_record_generator

```
hdl_record_generator string
```

Chooses a scheme to name individual bits of record ports and registers. The string argument must include `%s` to indicate the record name of the bus signal, and a second `%s` to indicate the field name. Set this global before performing `do_build_generic`. Related command are `do_blast_busses` and `do_rename`.

Default: `%s_%s`

hdl_resource_sharing

```
hdl_resource_sharing { true | false }
```

Lets the software collect information for sharing. Enter this command before entering the `do_build_generic` command. During the `do_optimize` phase, set the global resource sharing command to `false` to disable resource sharing, or it will attempt to reclaim area after timing optimizations.

Default: `true`

hdl_tree_height_reduction

```
hdl_tree_height_reduction { true | false }
```

Reduces the height of an expression tree by balancing its subtrees. THR improves performance by reducing the critical path. THR is done during the `do_build_generic` phase of the HDL synthesis flow.

Default: `true`

hdl_undriven_net_value

```
hdl_undriven_net_value { 0 | 1 | x | z | none}
```

Connects each undriven net in a module to the specified value unless the `none` value is specified. If the `none` value is specified, undriven nets remain unconnected. An undriven bit of a partially driven bus net is considered an undriven net.

Default: `0`

hdl_undriven_pin_value

```
hdl_undriven_pin_value { 0 | 1 | x | z | none }
```

Connects each undriven input pin in a module or cell instantiation to the specified value unless the `none` value is specified. If the `none` value is specified, undriven pins remain undriven. An undriven bit of a partially driven output port is considered an undriven output.

Default: `none`

hdl_undriven_port_value

```
hdl_undriven_port_value { 0 | 1 | x | z | none}
```

Connects each undriven output port in a module to the specified value unless the `none` value is specified. If the `none` value is specified, undriven ports remain unconnected.

Default: `none`

hdl_verilog_ignore_null_ports

```
hdl_verilog_ignore_null_ports { true | false }
```

When set to `true` the parser will ignore null Verilog port declarations and issue a warning. When set to `false`, null ports will be implemented and a warning will be issued. The implementation of null ports entails creating netlist ports with generated names that are unattached to anything within the module.

Default: `true`

hdl_verilog_out_columns

`hdl_verilog_out_columns` *`integer`*

Specifies the maximum line length for writing out Verilog netlist in files. Can be overridden by global `hdl_write_multi_line_port_maps`.

Default: `80`

hdl_verilog_out_compact

`hdl_verilog_out_compact { true | false }`

Specifies whether to write out compact files for Verilog netlist output. The compact files have multiple statements on one line. It may not be very readable. If this variable is set to `false`, only one statement is written per line.

Default: `true`

hdl_verilog_out_declare_implicit_wires

`hdl_verilog_out_declare_implicit_wires { true | false }`

Implicit wires in Verilog do not require a declaration. If set to `true` the declarations for implicit wires are also written.

Default: `false`

hdl_verilog_out_no_negative_index

`hdl_verilog_out_no_negative_index { true | false }`

Converts negative indices into positive indices for Verilog. This global bases all negative buses at zero and then counts up from zero.

Default: `false`

hdl_verilog_out_no_tri

`hdl_verilog_out_no_tri { true | false }`

Specifies whether three-state nets should be written out as wires in the Verilog netlist.

Default: `false`

hdl_verilog_out_prim

```
hdl_verilog_out_prim { true | false }
```

When set to true, primitive Verilog operators are written instead of the ATL equivalent components.

Default: `true`

hdl_verilog_out_source_track

```
hdl_verilog_out_source_track { true | false }
```

Keep track of the source code (RTL). In various reports, the tool attempts to identify the sections of code which caused particular effects on the design.

Default: `false`

hdl_verilog_out_unconnected_style

```
hdl_verilog_out_unconnected_style { none | partial | full }
```

Selects the netlisting style for unconnected instance pins.

- `full`

  ```
  C i0 (.UNUSED(UNCONNECTED_3));
  ```

- `partial`

  ```
  C i0 (.UNUSED());
  ```

- `none`

  ```
  COMP i0 ();
  ```

Default: none

hdl_verilog_out_use_supply

```
hdl_verilog_out_use_supply { true | false }
```

Specifies whether constant signals (1 or 0) are declared as supply signals (`supply1` or `supply0`). If this variable is set to `true`, the generated Verilog code will contain supply declarations. If it is set to `false` then the literal constants `1'b1` and `1'b0` are used for connection to power and ground.

Default: `false`

hdl_verilog_read_version

```
hdl_verilog_read_version { 1995 | 2001| dp }
```

Handles potential incompatibility by enabling verilog parsing for Verilog–1995, Verilog–2001, and Verilog-dp (datapath).

hdl_verilog_vpp_arg

```
hdl_verilog_vpp_arg string
```

Passes arguments to VPP (Verilog pre-processor). The typical argument passed is the search path. For example, if this variable is set to `-I/home/rtl`, `ac_shell` would search for Verilog files in `/home/rtl`. More than one option may be specified to this variable by creating one string with all the options to be used. The options within the string must be separated by one or more spaces. Valid options are as follows:

- `-Idirectory_path`

  Specifies the directory path.

- `-Dmacro=value`

  Equivalent to define macro value.

- `-Dmacro`

  Equivalent to define macro.

Note: `hdl_verilog_vpp_arg` replaces `vpp_arg`.

hdl_vhdl_case

```
hdl_vhdl_case { lower | upper | original }
```

Causes the VHDL analyzer to store VHDL identifiers and operators in lower case, upper case, or the case given in the source file.

Default: `original`

hdl_vhdl_environment

```
hdl_vhdl_environment { standard | synopsys| common | synergy}
```

Specifies the selection of the predefined arithmetic libraries.

Default: `common`

hdl_vhdl_lrm_compliance

```
hdl_vhdl_lrm_compliance { true | false }
```

When set to `true`, `read_vhdl` enforces a more strict interpretation of the VHDL LRM. This variable lets you verify that your VHDL code is compliant with the LRM, such that it is likely to work on other VHDL tools.

The following features are disallowed when set to `true`, but allowed when set to `false`:

- Treatment of a concatenation as a locally static expression. This allows constructs such as the following:

  ```
  case expr is
        when "001" & '1' => ...
  ```

- A name *X* can be used in the definition of a different *X*:

  ```
  constant c : integer := 3;
  function f (c : integer := c) is ...
  ```

- In VHDL-1987, a function call with globally static arguments is treated as a globally static expression. For instance:

  ```
  function f(x : integer) return integer;
  ...
  generic (y : integer := f(3));
  ```

- Initialization of interface object with function call.

Default: `false`

hdl_vhdl_preferred_architecture

`hdl_vhdl_preferred_architecture` *`name`*

Sets the name of preferred architecture to use with an entity when there are multiple architectures.

Default: `""`

hdl_vhdl_read_version

`hdl_vhdl_read_version { 1987 | 1993 }`

Specifies the VHDL version to be used when reading VHDL designs. If you change the `hdl_vhdl_read_version` read version at any time, the software automatically re-maps the IEEE and STD VHDL libraries to the correct directory in the software release. In this way they pick up the correct versions of standard packages in these libraries.

Default: `1993`

hdl_vhdl_reuse_units

`hdl_vhdl_reuse_units { true | false }`

Specifies whether `do_build_generic` should load all the analyzed units from a previous `ac_shell` session for the defined VHDL libraries.

Default: `false`

hdl_vhdl_write_architecture

`hdl_vhdl_write_architecture { true | false}`

Specifies whether to write VHDL architectures when `write_vhdl` is called.

Default: `true`

hdl_vhdl_write_architecture_name

`hdl_vhdl_write_architecture_name` *`architecture_name`*

Sets the name for architectures for all the modules during the VHDL netlist generation. The variable, *`architecture_name`*, can be any string and may contain at most one '%s'. Using `'%s'` causes a different name to be generated for each architecture in your design (for instance: `netlist_TOP`, `netlist_BOTTOM`).

Default: `netlist`

Note: Giving the global `hdl_vhdl_write_architecture_name` a value with any format specification having more than one '%s' results in an error. For example, *`netlist`*, *`%s`*, *`A_%s_B`* are all acceptable; *`A%s%s`* is not.

hdl_vhdl_write_bit_type

`hdl_vhdl_write_bit_type { std_logic | std_ulogic }`

Specifies whether the netlist contains `std_logic`/`std_logic_vector` ports or `std_ulogic`/`std_ulogic_vector` ports.

Default: `std_logic`

hdl_vhdl_write_components

`hdl_vhdl_write_components { true | false }`

Determines whether any component declarations for technology cells will be written out during VHDL netlisting.

Default: `true`

hdl_vhdl_write_entity

`hdl_vhdl_write_entity {true | false}`

Specifies whether to write VHDL entities when `write_vhdl` is called.

Default: `true`

hdl_vhdl_write_entity_name

`hdl_vhdl_write_entity_name` *`string`*

Sets the name for entity representing the current module during VHDL netlisting. For hierarchical designs, this variable only affects the name of the current top-level module, all descendant modules use their own names. If set to the empty string (""), the current module name is used as the entity name.

Default: ""

hdl_vhdl_write_packages

`hdl_vhdl_write_packages` *`lib1.pack_x | lib1 ...`*

Specifies the list of library and package pairs for which the VHDL netlister will write out library and use clauses before each module that is written out. Specifying only the library (omitting the package pair) is also permissible.

Default: `ieee.std_logic_1164`

hdl_vhdl_write_version

`hdl_vhdl_write_version { 1987 | 1993 }`

Specifies the VHDL version to be used for writing out VHDL netlists.

Default: `1993`

hdl_write_gnd_name

`hdl_write_gnd_name` *`string`*

Specifies a name to be used for ground net in the netlist.

Default: `AMBIT_GND`

hdl_write_multi_line_port_maps

`hdl_write_multi_line_port_maps { true | false }`

When set to `true`, a port map can span over more than one line. If set to `false`, a port map will always be written on one line, ignoring the `hdl_verilog_out_columns` limit.

Default: `true`

hdl_write_top_down

```
hdl_write_top_down {true | false}
```

Specifies whether the design hierarchy should be written out in a top-down or bottom-up fashion. When the global is set to `true`, higher-level modules precede the lower-level modules in the netlist. When the global is set to `false`, lower-level modules are written out prior to modules that instantiate them.

Default: `false`

hdl_write_vdd_name

```
hdl_write_vdd_name string
```

Specifies a name to be used for VDD net in the netlist.

Default: `AMBIT_VDD`

7

Low Power Synthesis Globals

power_analysis_over_count_factor

```
power_analysis_over_count_factor {true|false}
```

When set to `true`, subtracts a factor from the calculated transition density value to compensate for over-counting during the calculation.

For non-primary input and sequential cell output nets, which do not have a user-specified transition density, the tool calculates the transition density to be used for power calculation. However, while calculating the transition density the tool assumes that transition at the inputs of a gate can not occur simultaneously. This assumption leads to an over-count in transition density, as the two transitions which are occurring simultaneously may not occur at the output of the gate due to path and gate delay.

Default: `true`

power_clock_gate_decommit_in_do_opt

```
power_clock_gate_decommit_in_do_opt {true|false}
```

When set to `true`, evaluates the clock gating logic for decommitting in the `do_optimize` flow. You must have inserted clock-gating logic for this global to have an effect.

Default: `true`.

power_clock_gate_insert_in_do_opt

```
power_clock_gate_insert_in_do_opt {true|false}
```

When set to `true`, inserts clock gating logic in the do_optimize flow.

Default: `true`

power_clone_approximate_insertion_delay

```
power_clone_approximate_insertion_delay {true | false}
```

Evaluates a cloning move in terms of clock insertion delay. When set to `true`, a rough and fast estimation of clock insertion delay is done for a cloning move. When set to `false`, clock insertion delay for a cloning move is estimated more accurately at the cost of run time.

Default: `true`

power_clone_declone_in_do_opt

`power_clone_declone_in_do_opt {true | false}`

When set to `true`, explores possible power savings through cloning and merging gating cells (decloning) in the `do_optimize` flow.

Default: `true`

power_clone_insertion_delay_uncertainity

`power_clone_insertion_delay_uncertainity` *`float`*

Sets the threshold value for improvement in insertion delay during cloning. When set to 0.1, a cloning move is accepted only if the insertion delay after cloning improves atleast by 10 percent of the average of the insertion delay before and after cloning.

Default: 0.1

power_clone_min_register_area_fraction

`power_clone_min_register_area_fraction` *`float`*

Evaluates a cloning move. When set to 0.005, cloning of a clock-gating instance is accepted only if the area of the bounding box of the registers controlled by it is greater than .005 * total core area.

Default: .005

power_connect_auto_test_pin_only

`power_connect_auto_test_pin_only {true|false}`

Connects the test ports in the top-module with the test ports in the lower-module in the bottom-up synthesis flow, only if the test ports were automatically inserted by LPS. Set this global to `true` to connect manually inserted top module test ports to the auto-inserted lower-module test ports.

Default is `true`.

power_default_prob

`power_default_prob` *`float`*

Sets a default probability value for power estimation. For power estimation, probability and transition density values are needed for every net. If you did not specify values for every net, the tool uses the specified probability value for the primary inputs and sequential cell output nets.

Default: `0.5`

power_default_toggle_rate

`power_default_toggle_rate` *`float`*

Sets a default transition density value for power estimation. For power estimation, probability and transition density values are needed for every net. If you did not specify values for every net, the tool uses the specified transition density value for the primary inputs and sequential cell output nets.

You must specify the transition density (number of transitions/simulation period in nanoseconds) as a positive floating number. If you want the tool to calculate the transition density, leave the default at -2. The tool calculates the default transition density as follows:

1. Adds the transition density of the primary inputs and sequential cell outputs, on which you did specify the transition density values.
2. Finds the average transition density value and uses it as default transition density.
3. If none of the primary inputs and sequential cell outputs has user-defined transition density values, the tool uses 1.e-4 as the default transition density.

Default: -2 (indicates that the value will be calculated)

power_dump_all_tc

`power_dump_all_tc {true|false}`

When set to `false`, dumps the user-asserted values of probability and transition density on the nets to the database. When set to `true`, the tool dumps both the user-asserted and the tool-calculated values to the database. This prevents the tool from recalculating these values, if they were already calculated before dumping to the database. Moreover, the power numbers before and after reading the database will match exactly if this global is set.

Default: `true` (dumps the user-asserted and the tool-calculated values)

power_gatelevel_opt_in_do_opt

```
power_gatelevel_opt_in_do_opt {true|false}
```

When set to `true`, performs gate-level power optimization in the `do_optimize` flow.

Default: `true`

power_internal_power_scaling

```
power_internal_power_scaling {true|false}
```

When set to `true`, the `CAP_UNIT`—specified in the timing library—is scaled appropriately. For example if `CAP_UNIT` is 0.5pf, the unit is set to 0.1pf for deriving power unit. This global affects the internal cell power calculation.

Note: Do not use this global if you generated the TLF library using the syn2tlf utility with the `-scalepower` option.

Default: `false`

power_multiple_vt_flow

```
power_multiple_vt_flow {high_vt_library high_vt_library...}
```

Controls the libraries used during power optimization. If set, LPS uses the specified high threshold voltage libraries for mapping and low effort timing optimization. All libraries (including high and low threshold voltage libraries) are used for full timing and power optimization.

power_no_sleepmode_in_resource_sharing

```
power_no_sleepmode_in_resource_sharing {true|false}
```

When set to `true`, automatically removes any sleep mode logic which controls a resource shared module during the sleep mode committing phase.

The sleep mode logic is inserted during `do_build_generic`. Resource sharing does not happen until timing optimization but before sleep mode commitment which happens before placement. Therefore, it is possible that some datapath modules which are selected as sleep mode candidates become resource shared modules after resource sharing. In most cases, this will significantly reduce the potential achievable power savings by sleep mode transformation. Therefore, this global gives you the flexibility to ignore the sleep mode logic during the sleep mode commitment phase in the scenario described above.

Default: `false`

power_operating_corner

```
power_operating_corner {min | typ | max}
```

Specifies the library to be used for power calculation. You can read `typ, min` and `max` libraries. Each library has power models for each cell.

The tool does not know from which library power models should be used.

- If the global is set to `max`, the power models specified in the `max` library are used. If the `max` library is not specified, then the `typ` library is used, and if the `typ` library is not specified, then the `min` library is used.
- If the global is set to `min`, the power models specified in the `min` library are used. If the `min` library is not specified, then the `max` library is used, and if the `max` library is not specified, then the `typ` library is used.
- If the global is set to `typ`, the power models specified in the `typ` library are used. If the `typ` library is not specified, then the `max` library is used, and if the `max` library is not specified, then the `min` library is used.

Default: `typ`

power_opt_no_tcf

```
power_opt_no_tcf {true|false}
```

When set to `true`, performs all low power optimization transformations, even if no TCF file is available. In this case, the default switching activity values are used.

Default: `false`

power_root_gate_in_do_opt

```
power_root_gate_in_do_opt {true|false}
```

When set to `true`, performs root gating in the `do_optimize` flow. You must have inserted clock-gating logic to perform root gating.

Default: `true`

power_slewmode_for_power_analysis

```
power_slewmode_for_power_analysis {best|worst|avg}
```

Specifies which slew to use in the look-up tables of the power models.

The power models in the library are typically in the form of look-up tables. One of the axes of the look-up tables is the slew on an input/output pin. This slew can be worst case, best case, or average of these two values.

- If the global is set to `worst (best)`, the worst (best) case slew (which it got from the timing engine) is used for getting the power numbers from the power look-up tables.
- If the global is set to `avg`, the tool will get both the worst case and best case slew on a pin and average this value to get the required slew for getting power numbers from the look-up tables.

Default: `worst`

power_use_clock_frequency

```
power_use_clock_frequency {true|false}
```

Affects the calculation of the transition density of clock pins.

If set to `true`, LPS determines the transition density of clock pins as follows:

- If timing constraints are present the transition density for the clock pin is calculated as: 2/clock_period
- In the absence of timing constraints, the transition density for the clock pin is calculated as: 2 times the default transition density.

If set to `false`, LPS uses the default transition density.

Note: You can set the default transition density (which applies to data and clock pins) through the `power_default_toggle_rate` global. If this global is not set, LPS uses1.0e-4.

Default: `false`

8

Name Changing Globals

dcn_bus_allow_conversion

```
dcn_bus_allow_conversion { true | false }
```

Set to `true` to allow conversion of the bus. Used to set the conversion rules for the `do_change_name` command.

Default: `false`

dcn_bus_allowed

```
dcn_bus_allowed string
```

Specify list of all allowed characters for the bus. Used to set the conversion rules for the `do_change_name` command.

Default: `][0123456789abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ_`

dcn_bus_first_restricted

```
dcn_bus_first_restricted string
```

Specify list of restricted first characters for the bus. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_bus_last_restricted

```
dcn_bus_last_restricted string
```

Specify list of restricted last characters for the bus. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_bus_map

`dcn_bus_map`

Maps all occurrences of a current bus name to a new bus name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_bus_max_length

`dcn_bus_max_length` *`integer`*

Specify maximum name length for the bus. Used to set the conversion rules for the `do_change_name` command.

Default: `32`

dcn_bus_prefix

`dcn_bus_prefix` *`string`*

Specify prefix characters to add to the bus name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_bus_remove_chars

`dcn_bus_remove_chars` *`string`*

Specify characters to remove from the bus name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_bus_replacement_char

`dcn_bus_replacement_char` *`char`*

Specify single replacement character to use in the bus. Used to set the conversion rules for the `do_change_name` command.

Default: B

dcn_bus_reserved_words

`dcn_bus_reserved_words` *`string`*

Specify string of reserved words to change for the bus. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_bus_restricted

`dcn_bus_restricted` *`string`*

Specify list of all restricted characters for the bus. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_inst_allow_conversion

`dcn_inst_allow_conversion { true | false }`

Set to `true` to allow conversion of the instance. Used to set the conversion rules for the `do_change_name` command.

Default: `false`

dcn_inst_allowed

`dcn_inst_allowed` *`string`*

Specify list of all allowed characters for the instance. Used to set the conversion rules for the `do_change_name` command.

Default: `][0123456789abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ_`

dcn_inst_first_restricted

`dcn_inst_first_restricted` *`string`*

Specify list of restricted first characters for the instance. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_inst_last_restricted

`dcn_inst_last_restricted` *`string`*

Specify list of restricted last characters for the instance. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_inst_map

`dcn_inst_map`

Maps all occurrences of a current instance name to a new instance name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_inst_max_length

`dcn_inst_max_length` *`integer`*

Specify maximum name length for the instance. Used to set the conversion rules for the `do_change_name` command.

Default: `32`

dcn_inst_prefix

`dcn_inst_prefix` *`string`*

Specify prefix characters to add to the instance name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_inst_remove_chars

`dcn_inst_remove_chars` *`string`*

Specify characters to remove from the instance name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_inst_replacement_char

`dcn_inst_replacement_char` *`char`*

Specify single replacement character to use in the instance. Used to set the conversion rules for the `do_change_name` command.

Default: `I`

dcn_inst_reserved_words

`dcn_inst_reserved_words` *`string`*

Specify string of reserved words to change for the instance. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_inst_restricted

`dcn_inst_restricted` *`string`*

Specify list of all restricted characters for the instance. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_module_allow_conversion

`dcn_module_allow_conversion { true | false }`

Set to `true` to allow conversion of the module. Used to set the conversion rules for the `do_change_name` command.

Default: `false`

dcn_module_allowed

`dcn_module_allowed` *`string`*

Specify list of all allowed characters for the module. Used to set the conversion rules for the `do_change_name` command.

Default: `][0123456789abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ_`

dcn_module_first_restricted

`dcn_module_first_restricted` *`string`*

Specify list of restricted first characters for the module. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_module_last_restricted

`dcn_module_last_restricted` *`string`*

Specify list of restricted last characters for the module. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_module_map

`dcn_module_map` *`list of name mappings`*

Maps all occurrences of a current module name to a new module name. Used to set the conversion rules for the `do_change_name` command.

Default: no mappings

Example

The following statement renames the module `oldName` to `newName` and the module `oldName2` to `newName2`:

```
set_global dcn_module_map {{oldName newName} {oldName2 newName2}}
```

dcn_module_max_length

`dcn_module_max_length` *`integer`*

Specify maximum name length for the module. Used to set the conversion rules for the `do_change_name` command.

Default: `32`

dcn_module_prefix

`dcn_module_prefix` *`string`*

Specify prefix characters to add to the module name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_module_remove_chars

`dcn_module_remove_chars` *`string`*

Specify characters to remove from the module name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_module_replacement_char

`dcn_module_replacement_char` *`char`*

Specify single replacement character to use in the module. Used to set the conversion rules for the `do_change_name` command.

Default: `M`

dcn_module_reserved_words

`dcn_module_reserved_words` *`string`*

Specify string of reserved words to change for the module. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_module_restricted

`dcn_module_restricted` *`string`*

Specify list of all restricted characters for the module. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_net_allow_conversion

`dcn_net_allow_conversion { true | false }`

Set to `true` to allow conversion of the net. Used to set the conversion rules for the `do_change_name` command.

Default: `false`

dcn_net_allowed

`dcn_net_allowed` *`string`*

Specify list of all allowed characters for the net. Used to set the conversion rules for the `do_change_name` command.

Default: `][0123456789abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ_`

dcn_net_first_restricted

`dcn_net_first_restricted` *`string`*

Specify list of restricted first characters for the net. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_net_last_restricted

`dcn_net_last_restricted` *`string`*

Specify list of restricted last characters for the net. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_net_map

`dcn_net_map`

Maps all occurrences of a current net name to a new net name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_net_max_length

`dcn_net_max_length` *`integer`*

Specify maximum name length for the net. Used to set the conversion rules for the `do_change_name` command.

Default: `32`

dcn_net_prefix

`dcn_net_prefix` *`string`*

Specify prefix characters to add to the net name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_net_remove_chars

`dcn_net_remove_chars` *`string`*

Specify characters to remove from the net name. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_net_replacement_char

`dcn_net_replacement_char` *`char`*

Specify single replacement character to use in the net. Used to set the conversion rules for the `do_change_name` command.

Default: `N`

dcn_net_reserved_words

`dcn_net_reserved_words` *`string`*

Specify string of reserved words to change for the net. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_net_restricted

`dcn_net_restricted` *`string`*

Specify list of all restricted characters for the net. Used to set the conversion rules for the `do_change_name` command.

Default: none

dcn_port_allow_conversion

`dcn_port_allow_conversion { true | false }`

Set to `true` to allow conversion of the port. Used to set the conversion rules for the `do_change_name` command.

Default: `false`

dcn_port_allowed

`dcn_port_allowed` *`string`*

Specify list of all allowed characters for the port. Used to set the conversion rules for the `do_change_name` command.

Default: `][0123456789abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ_`

dcn_port_first_restricted

`dcn_port_first_restricted` *`string`*

Specify list of restricted first characters for the port. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_port_last_restricted

`dcn_port_last_restricted` *`string`*

Specify list of restricted last characters for the port. Used to set the conversion rules for the `do_change_name` command.

Default: `$`

dcn_port_max_length

`dcn_port_max_length` *`integer`*

Specify maximum name length for the port. Used to set the conversion rules for the `do_change_name` command.

Default: `32`

dcn_port_map

`dcn_port_map`

Maps all occurrences of a current port name to a new port name. Used to set the conversion rules for the do_change_name command.

Default: none

dcn_port_prefix

`dcn_port_prefix` *`string`*

Specify prefix characters to add to the port name. Used to set the conversion rules for the do_change_name command.

Default: none

dcn_port_remove_chars

`dcn_port_remove_chars` *`string`*

Specify characters to remove from the port name. Used to set the conversion rules for the do_change_name command.

Default: none

dcn_port_replacement_char

`dcn_port_replacement_char` *`char`*

Specify single replacement character to use in the port. Used to set the conversion rules for the do_change_name command.

Default: P

dcn_port_reserved_words

`dcn_port_reserved_words` *`string`*

Specify string of reserved words to change for the port. Used to set the conversion rules for the do_change_name command.

Default: none

dcn_port_restricted

`dcn_port_restricted` *`string`*

Specify list of all restricted characters for the port. Used to set the conversion rules for the `do_change_name` command.

Default: none

9

PKS Globals

auto_block_rc_rule

```
auto_block_rc_rule {true | false}
```

When set to `true`, this global is equivalent to using the `set_block_rc_rule -auto -all_blocks` command. If set to `false`, the block rc rules are not automatically set for all blocks. You must manually set the block rc rules for specific block instances using the `auto_block_rc_rule` command.

Default: `true`

drv_failure_verbosity

```
drv_failure_verbosity {0 | 1}
```

When set to `1`, gives feedback at the end of DRV fixing about why certain DRVs were not fixed. For each net on which the tool fails to fix DRVs, prints a message that includes information about which buffering and resizing moves where attempted, and the reasons they failed.

Default: `0`

enable_re_placement

```
enable_re_placement {true | false}
```

Setting this global to `true` enables the re-placement transform during post-placement optimization. Re-placement changes the locations of instances along the critical path in an effort to improve slack. This transform is useful when instances (particularly flip-flops) have been placed in locations that unnecessarily extend path length.

When the clock propagation mode is propagated, re-placement will move instances that are not associated with clocks.

Default: `true`

infer_unplaced_physical_pins

```
infer_unplaced_physical_pins {on | off}
```

Prevents automatic assignment of placement and layer to any physical pins found in the DEF that came into the DEF that way, when this global is set to on. You might want to use this for blocks with power connections already in the form of rings, where adding an extra pin would cause routing violations.

Default: `off`

Example

If your input DEF has a line as follows:

```
- VDD + NET VDD + DIRECTION INOUT + USE POWER ;
```

Setting the `infer_unplaced_physical_pins` to on means that the output DEF will remain the same for this pin. If the global is left as its default setting (`off`), the tool assigns a placement and layer, making the output DEF something like the following:

```
- VDD + NET VDD + DIRECTION INOUT + USE POWER + LAYER METAL2 (-100 0) (100 1120) +
PLACED (685695 1370370) S ;
```

macro_directional_blockage

```
macro_directional_blockage {true | false}
```

When `pks_directional_blockage` is set to `true`, `macro_directional_blockage` controls whether a blockage box blocked in the horizontal or vertical direction only will be regarded as a blockage at all. When `macro_directional_blockage` is set to `true` but the directional blockage is not from a macro cell (for instance, when it is created by the `create_blockage` command on all horizontal layers), then this global is ignored by the Steiner router.

Default: `true`

module_boundary_optimization

```
module_boundary_optimization {true | false}
```

By default, BuildGates Extreme and PKS do module boundary optimization after uniquification. This means that if it helps simplify the logic inside the module, logic polarity at a port may be changed, or some nets or connectivity may be removed or optimized away at

the hierarchical instance boundary. To disable this behavior, set the `module_boundary_optimization` global to `false`.

Default: true

pks_cell_blockage_min_size

```
pks_cell_blockage_min_size integer_value
```

When the `pks_directional_blockage` global is set to 2, this global is used to derive a minimum size (`min_size = pks_cell_blockage_min_size*minPitch`). When constructing blockages from macro cell LEF-defined obstructions, any LEF obstruction with a width or height less than the minimum size is ignored. The default is `0`, which means the minimum size is derived from the cell object's size statistics and will be less than `10*minPitch`. This global can be regarded as a macro blockage resolution.

Default: `0`

pks_def_route_conversion

```
pks_def_route_conversion [wroute | native]
```

Default: `wroute`

Controls how the routes read in through DEF are converted into the internal route graph representation stored in the database.

Use the `wroute` option to use WRoute for this conversion.

Use the `native` option to use PKS code for this conversion and to reduce the increased peak memory usage associated with WRoute.

pks_directional_blockage

```
pks_directional_blockage {0 | 1 | 2}
```

Allows you to specify how macro blocks are to be handled. The possible values are as follows:

Value	Description
0	A blockage has *no* routable direction (directionless blockage). A macro block is considered a directionless blockage only when the porosities in both the horizontal and vertical directions are less than the `min_porosity` in each direction. In this mode, only directionless blockages are considered.
1 (default)	A blockage may have one routable direction (directional blockage). A macro block is considered a directional blockage when one and only one of the porosities is less than the `min_porosity` for the direction. In this mode, both directionless and directional blockages are considered.
2	The only difference between this mode and the previous mode is that a macro is divided into a group of smaller directionless or directional blockages based on cell obstructions in the LEF. In this mode, the concept of porosity becomes irrelevant. It is intended to find a more optimal route which can go along a feedthrough channel within a macro block. Use this mode to handle pad obstructions.

Default: `1`

pks_do_place_option

`pks_do_place_option` *`option`*

Specifies a quoted string of command line options supported by `do_place`.

`do_place` recognizes the value of `pks_do_place_option` as its command line option. Present `do_place` command line options take precedence should any conflict arise (PKS specific).

Default: " "

Example

If the variable is set to `-net_weight netA netB 5` and the command line option is `-net_weight netB netC 10`, it is equivalent to the command line option `-net_weight netA 5 -net_weight netB netC 10` without setting the variable.

pks_do_route_option

`pks_do_route_option` *`option`*

Specifies a quoted string of command line options supported by `do_route`.

`do_route` recognizes the value of `pks_do_route_option` as its command line option. Present `do_route` command line options take precedence should any conflict arise (PKS specific).

Default: `" "`

pks_encounter_exe

`pks_encounter_exe` *`encounter_path`*

Points to the Encounter™ executable to be used for legalization.

Default: `""` (If not set or set to `""`, the UNIX search PATH will be used to find an Encounter executable.)

pks_ignore_pad_net

`pks_ignore_pad_net {true | false}`

If set to `true`, PKS will treat any two pin net between a placed pad instance and a placed boundary pin as a zero RC connection. If this result is not desired, use this global to force wire delays to be computed on these nets.

Default: `true`

pks_legalization_technique

`pks_legalization_technique [batch_qp] [fe] [native] [qplace]`

Specifies a legalization technique.

Arguments and Options

`batch_qp`	Specifies that legalization be done using QPlace® called in batch mode. **Note**: Using this option requires a Qplace executable. You must use the `pks_qplace_exe` global to specify an executable for PKS to use.
`fe`	Specifies that legalization be done using the `refinePlace` command in Encounter™. When the `this` global is set to `fe`, PKS calls the Encounter executable in batch mode. Once the legalization completes, the final placement is read back into PKS through a DEF generated by Encounter. **Note**: Using this option requires an Encounter executable. You must use the `pks_encounter_exe` global to specify an executable for PKS to use.
`native`	Specifies that legalization be done using the native PKS legalization method.
`qplace`	Specifies that legalization be done using QPlace called from within PKS.

pks_pad_pin_attribute

```
pks_pad_pin_attribute attribute
```

If an instance pin has the attribute specified by this global, then the net is regarded as a pad net and is not buffered.

Default: `""`

pks_pass_user_rows_for_routing

```
pks_pass_user_rows_for_routing {true | false}
```

Instructs the tool to pass user-specified rows to `do_route` instead of computed rows.

Default: `false`

pks_qplace_exe

`pks_qplace_exe` *`qplace_path`*

Points to the QPlace executable to be used for legalization.

Default: `""` (If not set or set to `""`, the UNIX search PATH will be used to find a `qplace` executable.)

pks_qp_timing_number_round_off

`pks_qp_timing_number_round_off int`

Floating point variations across different platforms can affect placement quality significantly because of the placement algorithm's sensitivity to the timing budgets being passed to it. This global can help in some situations by reducing the number of significant digits in the timing calculations being used to drive placement. Set this global to the number of significant digits you wish to retain. If you set it to 0, no rounding is done.

Default: `0`

pks_respect_region_in_opt

`pks_respect_region_in_opt {true | false)`

When set to `true`, instructs the PKS optimization engine to *not* move cells in a region to a position outside that region. This global will be applied to all regions.

Note: Setting this global to `true` may affect the general quality of results.

Default: `false`

pks_snap_pin_locations

`pks_snap_pin_locations {0 | 1}`

Instructs the pin location generation function to snap the pin locations to the nearest track location when reading in the pin locations from a def or pdef, or from any other external source. (PKS specific)

Default: `0` (retain external pin locations)

pks_use_encounter_mode

```
pks_use_encounter_mode {on | off}
```

Instructs the tool to treat slashes (/) in a region or group name as part of the name and not a hierarchy delimiter.

Default: `off`

pks_use_flat_group_names_in_def

```
pks_use_flat_group_names_in_def {on | off}
```

Instructs the tool to treat slashes (/) in a group name as part of the name and not a hierarchy delimiter.

Default: `on`

pks_use_flat_region_names_in_def

```
pks_use_flat_region_names_in_def {on | off}
```

Instructs the tool to treat slashes (/) in a region name as part of the name and not a hierarchy delimiter.

Default: `on`

pks_wroute_exe

```
pks_wroute_exe wroute_path
```

Points to the `wroute` executable to be used with the `do_wroute` and `do_wroute_eco` commands.

Default: `""` (If not set or set to `""`, the UNIX search PATH will be used to find a `wroute` executable.)

steiner_route_in_io

```
steiner_route_in_io {true | false}
```

When set to `false`, the steiner router adds extra cost to any routes going inside the IO region (from the die box boundary to pad cell's innermost boundary) to discourage routing inside the region.

Default: `true`

10

Test Synthesis Globals

dft_alphabetical_scan_chain

```
dft_alphabetical_scan_chain {on | off}
```

When set to on, scan chains are ordered alphabetically.

Default: `off`

dft_allow_scan_path_inv

```
dft_allow_scan_path_inv {true | false}
```

Controls whether scan-data can be inverted along the scan path. Allows inversion in the scan path when set to `true`, prevents inversions when set to `false`. Inversions in the scan path can occur in two situations:

The scan register's Q-bar output is selected for the scan data to reduce loading on the Q output (see the global variable `dft_scan_output_pref min_load`).

A scan register was selected with a single, Q-bar, output pin.

When set to `false`, the scan insertion tool adds an inverter to the scan-data path immediately following an inversion. When set to `true`, any inversions in the scan data path are marked with an <i> in the scan chain report file.

Default: `true`

dft_allow_swapping_in_reordering

```
dft_allow_swapping_in_reordering {on|off}
```

When enabled, allows swapping of scan elements (flip-flops or fixed segments) between scan chains determined to be compatible during placement-based scan chain reordering. Compatibility is based on elements belonging to the same clock domain, as determined by the `check_dft_rules` command.

Note: At this time, this global affects only the scan connection operations that reorder existing scan chains. The affected commands are:

- `do_optimize -pks -skip_scan_configuration -scan_reorder`
- `do_xform_connect_scan -pks -preserve_config`
- `do_place -scan_reorder`

Default: `off`

dft_disconnect_scan_during_placement

```
dft_disconnect_scan_during_placement {on|off}
```

When enabled, disconnects the scan chains prior to placement, and restores the scan chains after placement.

Default: `off`

dft_enable_combinational_loop_check

```
dft_enable_combinational_loop_check {true | false}
```

When set to `true`, the `check_dft_rules` command reports any combinational feedback loops in the netlist. Alternatively, you can use the `check_netlist` command instead.

Default: `false`

dft_enable_race_condition_check

```
dft_enable_race_condition_check {true | false}
```

When set to `true`, the `check_dft_rules` command issues warnings for any flip-flop with a potential race condition between its data and clock signal.

Default: `false`

dft_fix_floating_net_violation

```
dft_fix_floating_net_violation [true | false]
```

Controls whether `do_xform_fix_dft_violations` has to fix floating asynchronous set and reset pin violations. Floating nets are generally regarded as design errors rather than DFT-rule violations. By default this type of DFT rule violation is not fixed. If you set this global to `true`, floating violations are fixed by connecting the set/reset pin to its inactive value both in test mode and in normal operation mode.

Default: `false`

dft_insert_terminal_lockup_element

```
dft_insert_terminal_lockup_element {on | off}
```

When enabled, configuration inserts a terminal lockup latch at the end of each scan chain connecting to the specified scan data output signal. This global has no effect when the scan chains are reordered using the `-preserve_config` mode.

Terminal lockup latch insertion is typically used for core modules which become embedded blocks in a higher level design. When incorporating blocks which contain terminal data lockup latches, the lower level scan chain segments must be first identified using `set_scan_chain_segment` assertion, prior to creating the top-level chains. See Inserting Terminal Data Lockup Latches for more information.

Default: `off`

dft_instance_name_prefix

```
dft_instance_name_prefix string
```

Specifies the prefix added to the instance names inserted by `do_xform_fix_dft_violations`, `do_xform_insert_testpoint`, and `do_xform_insert_shadow_dft`. The instance names are not guaranteed to persist after subsequent transformation steps, such as technology mapping or optimization.

Default: `BG_DFT_FIX`

dft_lockup_element_type

```
dft_lockup_element_type [level_sensitive | edge_sensitive]
```

Controls the type of device to be used for lockup elements when mixing different domains in a scan chain. You can choose between an edge-sensitive device (flip-flop) or a level-sensitive device (latch).

Default: `level_sensitive`

dft_min_scan_wire_length

`dft_min_scan_wire_length` *`distance`*

Specifies the minimum placement-based wire length to be used for scan data path connections during scan chain reordering. Specify the distance in units that are consistent with the technology library units read from the LEF file (units are typically specified in microns).

Use this global to minimize hold-time violations that can be created by a short wire length in the scan data path.

dft_scan_avoid_control_buffering

`dft_scan_avoid_control_buffering {true | false}`

Controls whether or not the tool buffers scan control signals. Scan control signals, such as scan-mode, tend to have high fanout. By default, the scan insertion tool buffers high fanout signals to avoid maximum signal loading conditions. Setting this variable to `true` prevents buffering of the scan control signal. For the muxed scan style, the affected scan control signal is the scan-mode signal.

- `true`

  The tool does not buffer any scan control signals. The scan signal buffering can be done after synthesis by an external tool.

- `false`

  The tool buffers the scan control signals which can result in extensive buffering of the scan mode port.

Default: `false`

dft_scan_enable_connect

`dft_scan_enable_connect { on | tieoff | floating }`

Controls how scan-enable pins are connected at each register.

- on

  Scan-enable pins are connected for normal scan operations throughout the hierarchy.

- `tieoff`

  Scan-enable pins are connected to the off state for each scan-register (low for most flip-flops, high for those with inverted enables).

- `floating`

  Scan-enable pins are not connected to anything. This value is recommended when using an external tool to complete the connection.

Default: `on`

dft_scan_output_pref

```
dft_scan_output_pref { non_inv | min_load }
```

Controls which scan register output, Q or Q-bar, is used for the scan data path.

- `non_inv`

  Specifies a preference for the non-inverted (Q) register output pin.

- `min_load`

  Specifies a preference for the register output pin with the smallest load. The `min_load` setting may decrease the load on the system data path and so minimize the impact of scan on the timing of the system data path. If desired, correct any scan data inversions occurring as a result of the `min_load` setting with the `set_global dft_allow_scan_path_inv` command. The Cadence® synthesis scan insertion tool can enforce the selected setting only on scan registers that have both a Q and a Q-bar output.

Default: `min_load`

dft_scan_path_connect

```
dft_scan_path_connect { chain | tieback | tie0 | tie1 | floating }
```

Controls how the scan chain is connected during a synthesis run. The recommended use model is to use `tieback` mode prior to the final synthesis run to avoid timing analysis on the scan chains. For the final synthesis run, use `chain` mode if the tool is connecting your scan chains or use `floating`, `tie0`, or `tie1` mode if an external tool is connecting your scan chains. Consult your external tool documentation to decide whether to use `floating`, `tie0`, or `tie1`. (For more details on using this variable, see the *Test Synthesis for BuildGates Synthesis and Cadence Physically Knowledgeable Synthesis (PKS)*.)

This global variable does not affect the scan register control connections. For example, scan-enable pins are always connected, (as per DFT global: dft_scan_enable_connect), regardless of the value set for this global variable.

- `chain`

  Instructs the scan insertion tool to connect the scan registers into scan chain(s)

- `tieback`

  Connects a scan register's scan data output pin to its own scan data input pin, emulating the loading effect on the scan data output without connecting the chain, so that optimization accurately reflects the loading that will occur once the scan chain is connected.

- `tie0`

  Connects the scan registers' scan data input pins to logic 0. If available, a logic 0 net or equivalent library cell is used. Otherwise a constant (`1'b0`) is used.

- `tie1`

  Connects the scan registers' scan data input pins to logic 1. If available, a logic 1net or equivalent library cell is used. Otherwise a constant (`1'b1`) is used.

- `floating`

  Leaves the scan registers' scan data input and output pins unconnected.

Default: `chain`

dft_scan_port_name_prefix

```
dft_scan_port_name_prefix prefixName
```

Changes the base names for scan-data input and output ports from `BG_scan` (default) to *`prefixName`*.

Default: null

dft_stop_analysis_at_complex_logic

```
dft_stop_analysis_at_complex_logic {on | off}
```

When importing a scan-mapped structural netlist that was synthesized using a third party tool (other than LogicVision), set this variable to `on`, if there is a possibility of logic gates such as AND/OR/MUX or any other single-output combinational logic gates residing in the scan-data

path (and not at the end of the scan chains for the purposes of shared-scan data output signals).

Example circuits are those with MBIST structures which use multiplexing logic to steer the scan data along the scan path. Even when these circuits are generally attributed with a `set_dont_touch_scan` property, you still need to specify `dft_stop_analysis_at_complex_logic` on, to correctly analyze the scan chains.

Default: `off`

For more information, see the Using dft_stop_analysis_at_complex_logic Global in *Design For Test (DFT) Using BuildGates Synthesis and Cadence PKS*.

dft_test_mode_port_name_prefix

`dft_test_mode_port_name_prefix` *`string`*

Specifies the prefix for test-mode ports inserted automatically by the tool to disable clock gating logic for Low-Power. Default: `TEST_MODE`

dft_use_dedicated_submodule_scan_ports

`dft_use_dedicated_submodule_scan_ports {on|off}`

Controls whether to insert dedicated scan ports during scan chain connection through lower submodule boundaries or whether to use functional ports whenever possible.

Use this global in the SoC Encounter flow when using partitioning which requires dedicated scan ports at the partition boundaries.

Default: `off`

dft_verbosity_level

`dft_verbosity_level` *`integer`*

When *`integer`* is set to a number greater than 1 the command `check_dft_rules` generates more detailed reports.

Default: `1`

11

Timing Globals

auto_slew_and_delay_degradation

```
auto_slew_and_delay_degradation {none | linear | exponential}
```

Lets you select which waveform model to use at the driver when computing the interconnect delay: A linear model which is a ramp and a single exponential model.

The waveform at the driver is created in such a way that the slews at the driver computed using library slew tables match the slews of the waveform computed using slew thresholds.

Default: `exponential`

auto_wire_load_selection

```
auto_wire_load_selection {true | false}
```

When set to `false`, disables the automatic selection of wire load models based on the area of hierarchical blocks containing the nets.

Default: `true`

cfpv_design_instance_name

`cfpv_design_instance_name` *`instance_name`*

Specifies the design instance name for the HDL output file. A path proven true using Critical False Path Verification (CFPV) analysis need not be true under the sequential operation of the circuit. To make sure that the path is true under sequential operation of the circuit, the sensitization condition of the path should be satisfied. CFPV writes out this sensitization condition in the form of an HDL simulation assertion module that can be plugged into dynamic simulation and monitored.

For more information, see the Reporting and Eliminating False Paths section in the *Timing Analysis Guide*.

cfpv_testbench_name

`cfpv_testbench_name` *`testbench_name`*

Specifies the HDL testbench for the HDL output file. A path proven true using Critical False Path Verification (CFPV) analysis need not be true under the sequential operation of the circuit. To make sure that the path is true under sequential operation of the circuit, the sensitization condition of the path should be satisfied. CFPV writes out this sensitization condition in the form of an HDL simulation assertion module that can be plugged into dynamic simulation and monitored.

For more information, see the Reporting and Eliminating False Paths section in the *Timing Analysis Guide*.

clock_gating_regardless_of_downstream_logic

```
clock_gating_regardless_of_downstream_logic {true | false}
```

Controls the treatment of a clock signal based on how it is used in the fanout cone of the clock-gating cell (down stream logic) when data and clock signals arrive at the inputs. A clock gating check will be performed at the clock gating cell based on whether or not the signal is expected at the fanout cone of the clock-gating cell.

When set to `true`, the clock phase will be propagated to all the pins in the fanout cone without respect to function or usage. This can cause optimization problems, since these pins are considered as part of the clock network. Clock networks are automatically set as `dont_modify` in PKS.

When set to `false`, signals are propagated as shown in Table 11-1 on page 152.

Normally, the output of a gate with a clock and data input fans out to other buffers or inverters in the clock network, or the gated clock output feeds register clock pins. In some cases, a gated clock output can be used as a data signal; for instance, it may be connected to a register data input. By default, timing analysis converts such clock signals to data signals automatically unless the global `clock_gating_regardless_of_downstream_logic` is set to `true`. In which case, the clock signal does not convert to a data signal.

In the special (rare) case where all of the downstream pins of a gated clock are data signals, timing analysis does not perform clock gating setup and hold checks. The arc from the data input is not blocked either. The clock signal, which is converted to data, as well as the data signal is propagated through the gate in this case.

Default: `false`

Tip

Determine whether a pin is clock or data using the following command:

`get_timing` *`pin_name`* `clkordata`

Table 11-1 Signal Treatment

Signal Expected	Clock Gating Check	Result
Clock—gated signal only connects to the clock pins of registers/latches	yes	Clock flows through gate; data is blocked.
Data—clock gating cell connects to data/preset/clear pins of registers/latches	No	Clock signal is converted to data. Both converted clock and data signals flow through gate and take part in down stream timing checks (if any). The entire fanout cone is considered as data logic and optimization engine optimizes it to meet timing constraints.
Data and clock	Yes	Data signal is blocked and clock flows through. For signal from a pin: If the gated signal is only used as data down stream, then the clock signal is converted to data and the optimization works on the logic. If the gated signal is used down stream as clock or as both clock or data, the signal continues as clock.

clock_gating_to_be_checked

```
clock_gating_to_be_checked {true | false}
```

When set to `true`, automatically checks whether both clock and data signals are connected to any gate. When clock and data meet, the timing of the data input is propagated to the clock input of the gate to ensure that there are no glitches.

Default: `true` (Cadence does not recommend `false`.)

Note: Setting this global to `false` only disables the automatic clock gating check introduced by the timing engine (reported as `ClockGatingSetup`/`ClockGatingHold`) check, which is the non-sequential setup (or hold) check defined in the library.

dcl_add_pincap_to_rlc

```
dcl_add_pincap_to_rcl {true | false}
```

When using an OLA library that is able to compute a reduced model of a parasitics network, the tool will rely on the OLA library to compute the reduced parasitics model. Using this global provides the flexibility to include or exclude the pin capacitance attached to the parasitics network that the tool passes to the OLA library.

This flexibility is needed because OLA supports a complex way to describe the parasitics network of each pin that is attached to the net, which can be specified in term of pin admittance. For OLA libraries that do not support such a model, set the global to `true` and the tool adds the pin capacitance to the parasitics network.

Default: `false`

dcl_debug_mode

```
dcl_debug_mode {true | false}
```

Sets the debug mode to `true` or `false`, depending on the OLA library. Some OLA libraries are compiled with debug on. These libraries will report a trace of API calls and also report each passed argument when this global variable is set to `true`.

Default: `false` (debug mode off, debug messages suppressed)

dcl_message_verbosity_level

```
dcl_message_verbosity_level integer
```

When the global `dcl_debug_mode` is `true` (on), this command sets the level of verbosity. The higher the level, the more debug messages are reported. The number of levels available depends on the OLA library developer.

Default: `0`

dcl_rcl_use_cellpin_name

```
dcl_rcl_use_cellpin_name {true | false}
```

If you are using the LSI OLA library, set this global to `true`. When set to `false` (default), BuildGates Synthesis will pass the hierarchical pin name while passing parasitics data to the OLA library for parasitics reduction. When set to `true`, BuildGates Synthesis will pass a cell-pin name only to the OLA library.

Passing the hierarchical pin name is the default behavior (`false`) because the OLA library can easily distinguish the nodename in the parasitics data.

Default: `false`

dcl_spec_1_4_use_pinname_for_pinprop

```
dcl_spec_1_4_use_pinname_for_pinprop {true | false}
```

Set this global to `true` if the OLA library you are using implements OLA specs 1.4 or below. The difference between the current OLA standard and of that in later specs, is that BuildGates Synthesis expects a pin pointer as an argument to an OLA function `clal`, whereas in older specs, a pin name string is expected in the argument. This global function is to support an OLA library that still implements the older specs.

Default: `false`

dcl_use_ssm_library

```
dcl_use_ssm_library {true | false}
```

Set this global to `true` if you are using the Silicon Metrics SSM (Silicon Smart Model) library.

Default: `false`

disable_pulsewidth_checks_on_data_pins

```
disable_pulsewidth_checks_on_data_pins
```

When set to `true` disables pulse width checks on pins to which a clock signal does not propagate.

Default: `false`

generated_clocks_inherit_ideal_latency

```
generated_clocks_inherit_ideal_latency {true | false}
```

When set to `false`, disables inheritance of clock insertion delay (latency) by generated clocks during ideal propagated mode. For more information, see Insertion Delays for Generated Clocks in the *Common Timing Engine (CTE) User Guide*.

Default: `true` (insertion delay is inherited)

generated_clocks_inherit_uncertainty

```
generated_clocks_inherit_uncertainty {true | false}
```

When set to `false`, disables inheritance of clock uncertainty by generated clocks. For more information, see Uncertainty for Generated Clocks in the *Common Timing Engine (CTE) User Guide*.

Default: `true` (uncertainty is inherited)

generated_clocks_scale_edges

```
generated_clocks_scale_edges {true | false}
```

Controls the way the generated clock parameters are calculated when the derivation method is `-multiply_by` or `-divide_by` with a division factor that is not a power of two.

Note: Once a generated clock is created by the system, after using a command such as `report_timing`, changing the value of the global `generated_clocks_scale_edges` does not result in an automatic update of the generated clock waveform. Generated clocks created after changing the global reflect the new value of the global. Set this global in the beginning of the session if its default value needs to be changed.

Default: `false`

latch_time_borrow_mode

```
latch_time_borrow_mode {budget | max_borrow}
```

Controls the mode for latch analysis. When set to `max_borrow`, the time borrowed is not equalized, which is similar to the Synopsys method. When set to `budget`, the slack is balanced on both sides of the latch. For more information, see Analyzing Latch-Based Designs in the *Common Timing Engine (CTE) User Guide*.

Default: `max_borrow`

lib_build_asynch_arc

```
lib_build_asynch_arc { true | false }
```

Creates delay arcs from set and reset pins to output pins of sequential library cells. Controls the creation of preset and clear to output arcs on sequential cells in the library.

Default: `false`

The `lib_build_asynch_arc` global controls if any asynchronous arcs in the timing library file are recognized, while the `lib_build_asynch_de_assert_arc` global controls whether the de-assert arcs are recognized or just the assert arcs.

If the `lib_build_asynch_arc` global and the `lib_build_asynch_de_assert_arc` globals are set to `true`, all asynchronous arcs are recognized.

If the `lib_build_asynch_arc` global is set to `true` and the `lib_build_asynch_de_assert_arc` global is set to `false`, only assert asynchronous arcs are recognized.

lib_build_asynch_de_assert_arc

```
lib_build_asynch_de_assert_arc { true | false }
```

Affects how set and clear de-asserts edges and controls whether the de-assertion edges are retained or not. These edges correspond to the removal of an asynchronous set or clear pin. These edges often have no effect and were ignored prior to s003.

Default: `false` (de-assertion edges will be removed)

If the `lib_build_asynch_arc` global and the `lib_build_asynch_de_assert_arc` globals are set to `true`, all asynchronous arcs are recognized.

If the lib_build_asynch_arc global is set to `true` and the `lib_build_asynch_de_assert_arc` global is set to `false`, only assert asynchronous arcs are recognized.

Examples

- The removal of a set or clear pin while both are asserted.
- The removal of a set pin on a transparent latch while the data input is low

lib_build_timing_cond_default_arc

```
lib_build_timing_cond_default_arc {true | false}
```

Controls the creation of the default timing arc. When set to `true`, the default timing arc is created from the default timing arc delays specified in the library. The default timing arc does not have the WHEN condition. When set to `false`, the default timing arc from the library is disabled.

Default: `true`

lib_cell_thresholds_for_reporting

```
lib_cell_thresholds_for_reporting {enable | disable}
```

When set to `disable`, uses the single design level thresholds for reporting. The same thresholds (upper and lower) apply to all cells in all libraries used in the design.

When set to `enable`, uses the individual library and cell level thresholds for reporting. This was the default behavior prior to the 4.0.9 release.

Default: `disable`

lib_ignore_pad_area

```
lib_ignore_pad_area {true | false}
```

When set to `true`, discounts the area of all pad cells. The wire load selection is based on the discounted area.

Default: `false`

max_slew_time_limit

```
max_slew_time_limit float
```

Specifies maximum slew limit for all cells in the library. The tool uses the worse (tighter) of the two limits set in the technology library and set by this variable.

Default: no limit — infinite

min_slew_time_limit

```
min_slew_time_limit float
```

Specifies minimum slew limit for all cells in the library.

Default: none

pvt_early_path

```
pvt_early_path { min | typ | max }
```

Associates a `min`, `typ`, or `max` PVT corner with early paths used to calculate `min`, `typ`, or `max` delay and slew values from the library. Delays and slews of gates and interconnects on early paths are calculated by selecting the corresponding values from the library.

Default: `max`

pvt_late_path

```
pvt_late_path { min | typ | max }
```

Associates a `min`, `typ` or `max` PVT corner with late paths used to calculate `min` or `max` delay and slew values from the library. Delays and slews of gates and interconnects on late paths are calculated by selecting the corresponding values from the library.

Default: `max`

report_precision

```
report_precision integer
```

Controls the number of digits appearing after the decimal point in the `report_timing` and `report_cell_instance_timing` timing reports.

Default: 2

report_timing_format

```
report_timing_format
```

Specifies a `report_timing` format. See the `report_timing` command `-format` option for more information.

For example, the following command tells `report_timing` to only list the hierarchical pin and arrival times in the report:

```
set_global report_timing_format {hpin arrival}
```

sdc_write_unambiguous_names

```
sdc_write_unambiguous_names {true | false}
```

When set to `true`, and if the SDC file version is 1.2 or newer, the `write_sdc` command writes out the sdc file using the -`hsc` option with @ as the hierarchical divider. The -`hsc` option

is used when an ambiguous name is encountered with a `get_*` list command during a `write_sdc` step. For example, `get_pins -hsc @` or `get_ports -hsc @`.

An ambiguous name is generally a flat name of a pin or port, which needs to use the escaped Verilog syntax in PKS.

Default: `true`

slew_limit

`slew_limit` *`float`*

This variable has been replaced with the `max_slew_time_limit` variable.

slew_propagation_mode

`slew_propagation_mode { worst_slew | critical_slew | fast }`

Specifies the mode of the delay calculation in the timing analyzer. There are three modes of delay calculation in the timing analyzer that differ in their treatment of slew calculation and propagation. The `fast` mode refers to a fast algorithm for slew propagation, which sacrifices some accuracy. The `worst_slew` mode refers to an accurate delay calculation algorithm, which propagates the worst of all inputs. The `critical_slew` mode refers to an accurate delay calculation algorithm, which propagates the slew in the critical inputs. This switch does not affect the mode of the timing analyzer during optimizations, which is always `fast`.

Default: `worst_slew`

slew_time_limit

`slew_time_limit` *`float`*

This variable has been replaced with the variable `max_slew_time_limit`.

target_technology

`target_technology` *`lib_x`* `...`

Sets the target technology library specified by *`lib_x`*. Sets the search path for the library, which must be a global so that it can be reset from the shell. Multiple target libraries are allowed. The tool default values such as the units for time and capacitance used in reports and assertions, the slew threshold definitions for specifying slew at ports, and the default operating conditions are picked up from the first library.

For the `get_global` command, this variable returns the current list of target libraries. The target technology libraries are searched in the following order:

- Cell Definitions

  Target libraries are searched sequentially in the order provided. Cells are searched for by name. Search is terminated when a match is found. If two cells have the same name in the libraries, only the first one found is identified.

- k-factor Derating

  For the parameters listed below, the first occurrence in the library list is set as the default value. Values are selected from the library which contains the parameter (for instance, wireload model). For any parameter, the following values are needed to compute the operating conditions based on derating:

  - The corresponding k-factor (`k_process_wire_cap`, `k_volt_wire_res`, and so on). Values are selected from the library which contains the parameter (for instance, wireload model).
  - The base or nominal operating condition values at which the parameter was characterized. Values are selected from the library which contains the parameter (for instance, wireload model).
  - The working operating condition values at which the design is run. The value is selected from the library which contains the design operating condition.

- List

  For instance: cell drive, cell pin, load, dcl function, mode array.

timing_allow_register_output_as_delay_through

```
timing_allow_register_output_as_delay_through { true | false }
```

When set to `true`, the output pins of registers are considered to be sequential start points for `set_path_delay_constraint`. If `false`, the output pins are considered combinational start points. Set to `false` for compatibility with Synopsys.

Note: Set this global prior to applying the `set_path_delay_constraint` command, which only affects constraints applied after the global has changed.

Default: `false`

timing_analysis_type

`timing_analysis_type {min_max | bc_wc}`

Switches the timing analysis from `min` and `max` analysis to best case and worst case. For detailed information on analysis, refer to "On and Off Chip Variation Analysis" in the *Common Timing Engine (CTE) User Guide*.

Default: `min_max`

timing_case_analysis_for_sequential_propagation

`timing_case_analysis_for_sequential_propagation {true | false}`

When set to `true`, calculates constants on the outputs of sequential elements. Each time you change the setting of this global, a complete timing update is required and incremental updates are abandoned. If the outputs evaluate to a constant, check arcs and delay arcs are disabled regardless of the setting.

Default: `false`

timing_commands_ignore_null_pin_list

`timing_commands_ignore_null_pin_list {true | false }`

When set to `true`, forces BuildGates Synthesis and PKS to continue when a constraint is processed with an invalid pin. By default, commands will return an error when an invalid pin is supplied.

Default: `false`

timing_cppr_threshold_ps

`timing_cppr_threshold_ps` *`float`*

Specifies the maximum amount of pessimism that CPPR analysis is allowed to leave in the report. The unit is pico seconds. Setting this global to a specified value means that all the paths with pessimism less than or equal to the specified value will be reported without having their pessimism removed. This global will not effect paths with pessimism greater than the specified value.

Default: 20 ps

timing_disable_bus_contention_check

```
timing_disable_bus_contention_check {true | false }
```

When set to `true`, disables propagation of maximum delay along three state disable timing arcs and minimum delay along three state enable arcs. If you know that bus contention conditions do not occur in your design, disable these checks by setting these globals to `true`.

Checks for setup and hold violations in three-state bus designs. Checks the worst delay path to ensure that the latched signals are stable and not unknown values - X. Ensures proper timing checks for bus contention.

Default: `false`

timing_disable_checkarcs_due_to_constant

```
timing_disable_bus_checkarcs_due_to_constant {true | false }
```

When set to `true`, disables check arcs to data pins when the output of either a gating clock or a sequential cell evaluates to a constant value.

Default: `true`

timing_disable_clockperiod_checks

```
timing_disable_clockperiod_checks {true | false}
```

When set to `true`, disables clock period checks in the timing model at the beginning of optimization. This global returns to its previous value after optimization.

Default: `false` (checks performed)

timing_disable_floating_bus_check

```
timing_disable_floating_bus_check {true | false}
```

When set to `true`, disables propagation of minimum delay along three state disable timing arcs and maximum delay along three state enable arcs. These checks are only valid during floating bus conditions.

Default: `false`

timing_disable_internal_inout_net_arcs

```
timing_disable_internal_inout_net_arcs {true | false}
```

When set to `true`, this global disables internal bidirectional feedback paths that span multiple instances.When set to `true`, this global enables internal bidirectional feedback paths, which are feedback paths that you may not want to use during analysis and optimization.

This global applies only to paths that span multiple instances. The global has no impact on internal feedback paths that are completely contained in one instance.

This global also exists in PrimeTime.

Default: `false`

For more information and examples, see Disabling and Enabling Internal Feedback Paths Involving Bidirectional Pins in the *Common Timing Engine (CTE) User Guide*.

timing_disable_internal_inout_cell_paths

```
timing_disable_internal_inout_cell_paths {true | false}
```

Default: `true`

When set to `false`, enables internal bidirectional feedback paths that are completely contained in one instance.

This global also exists in PrimeTime and makes the global `bidi_io_arc` obsolete. The semantics of the `timing_disable_internal_inout_cell_paths` global is the opposite of the `bidi_io_arc` global. For example:

```
set_global timing_disable_internal_inout_cell_paths true
```

is equivalent to

```
set_global bidi_io_arc false
```

For more information and examples, see Disabling and Enabling Internal Feedback Paths Involving Bidirectional Pins in the *Common Timing Engine (CTE) User Guide*.

timing_disable_nochange_checks

```
timing_disable_nochange_checks {true | false}
```

When set to `true`, disables no change checks in the timing model.

Default: `false` (checks performed)

timing_disable_pulsewidth_checks

```
timing_disable_pulsewidth_checks {true | false}
```

When set to `true`, disables pulse width checks in the timing model at the beginning of optimization. The global returns to its previous value after optimization.

Default: `false` (checks performed)

timing_disable_recovery_removal_checks

```
timing_disable_recovery_removal_checks {true | false}
```

When set to `true`, disables recovery and removal checks in the timing model.

Default: `false` (checks performed)

timing_disable_skew_checks

```
timing_disable_skew_checks {true | false}
```

When set to `true`, disables skew checks in the timing mode.

Default: `false`

timing_disable_test_signal_arc

```
timing_disable_test_signal_arc {true | false}
```

When set to `true`, timing analysis will not analyze the signal arcs coming from or going to the test pin. Such pins are marked in the timing library with either the `test_scan_in` or the `test_scan_enable` attribute for the input pin or the `test_output_only` attribute for the output pin.

Default: `false`

The following are the equivalent TLF pin attributes:

Type	Liberty	TLF
enable	`test_scan_enable`	`SCAN_PINTYPE` (ENABLE)
input	`test_scan_in`	`SCAN_PINTYPE` (INPUT)

Type	Liberty	TLF
output	`test_output_only`	`SCAN_PINTYPE` (OUTPUT)
	`test_output_only`	`DEFINE_ATTRIBUTE` (TEST OUTPUT ONLY)

timing_driven_cong_analysis

```
timing_driven_cong_analysis {0 | 1}
```

When set to `1`, enables a timing-driven global route when doing a wroute global route based congestion analysis (PKS specific).

Default: `0` (timing driven mode is turned off)

timing_ignore_slew_for_disable_arcs

```
timing_ignore_slew_for_disable_arcs {true | false}
```

When set to `true`, ignores slew from tri-state disable arcs, so that the slew will not be used for timing analysis or for checking design rules.

Default: `false`

timing_ignore_when_start_end

```
timing_ignore_when_start_end { on | off}
```

Default: `off`

Affects how timing checks are evaluated with respect to conditional expressions involving when (`COND` in TLF), `when_start` (`COND_START` in TLF) and `when_end` (`COND_END`). If this global is set to `off` and the when expression is missing, a new when expression will be derived as follows:

- new `when` becomes `when_start` if `when_start` is present but `when_end` is not
- new `when` becomes `when_end` if `when_end` is present but `when_start` is not
- new `when` becomes a conjunction of `when_start` and `when_end` when both `when_start` and `when_end` are present

For example, consider the following setup checks:

```
SETUP( D1 => CK COND_START( CK == 1'b0 ) SDF_COND_START( CK == 1'b0 )
```

```
    01 POSEDGE timingCheckModel1 )

SETUP( D2 => CK COND_END( CL == 1'b1 ) SDF_COND_END( CL == 1'b1 )
    01 POSEDGE timingCheckModel2 )

SETUP( D3 => CK COND_START( CK == 1'b0 ) SDF_COND_START( CK == 1'b0 )
    COND_END( CL == 1'b1 ) SDF_COND_END( CL == 1'b1 )
    01 POSEDGE timingCheckModel3 )
```

The above setup checks will be evaluated during timing analysis as if they were specified as:

```
SETUP( D1 => CK COND( CK == 1'b0 ) 01 POSEDGE timingCheckModel1 )
SETUP( D2 => CK COND( CL == 1'b1 ) 01 POSEDGE timingCheckModel2 )
SETUP( D3 => CK COND( ( CK == 1'b0 ) &&  ( CL == 1'b1 ) )
       01 POSEDGE timingCheckModel3 )
```

If the `timing_ignore_when_start_end` is on, `when_start` and `when_end` expressions will not come into play during timing analysis. This leads to more active timing checks, and hence, more pessimistic analysis. No warning messages will be issued for `when_start` and `when_end` expressions that are ignored.

timing_parasitics_delay_model

```
timing_parasitics_delay_model [elmore | PandR]
```

Controls the type of the reduced models generated from detailed parasitics. Detailed parasitics are loaded in PKS using the `read_spf` command or the `read_spef` command.

Default: `PandR`

- elmore

  Specifies that the reduced model is a PI model for the driver load and specifies elmore for the interconnect delay. This model is simple and accurate enough for most nets.

- PandR

  Specifies that the reduced model is a PI model for the driver load and specifies poles and residues (PandR) for the interconnect delay. This model is used only for critical nets (nets with significant wire resistance compared to the driver resistance). Using poles and residues leads to more accurate delay at the expense of runtime and memory consumption.

timing_reduce_multi_drive_net_arcs

```
timing_reduce_multi_drive_net_arcs {true | false}
```

Controls the reduction of the number of net arcs created for timing analysis for nets driven by parallel buffers. If a net is driven by n parallel buffers and is connected to `m` sink pins, the total number of net arcs created is `n*m`. For example, if n is `100` and `m` is `20000` as is the case in some clock network schemes, the number of net arcs created for timing analysis is 2 million. This results in a runtime and memory penalty.

This global lets you reduce the number of net arcs created. When the global is set, net arcs are only created from one driver. In the example, the number of net arcs will be `20000`.

Default: `false`

timing_reduce_multi_drive_net_arcs_threshold

`timing_reduce_multi_drive_net_arcs_threshold` *`int`*

Sets a threshold number used by the tool to trigger the reduction of timing arcs of nets driven by parallel buffers. The reduction takes place if the number of arcs is greater than the threshold value and if the timing_reduce_multi_drive_net_arcs global is set to `true`.

Default threshold value: `10000`

timing_remove_clock_reconvergence_pessimism

`timing_remove_clock_reconvergence_pessimism`

Enables CPPR (Common Path Pessimism Removal) analysis during `report_timing`. When set to `true`, the `report_timing` command removes pessimism from slack calculation.

timing_self_loop_paths_no_skew

`timing_self_loop_paths_no_skew {true | false}`

When set to `true`, factors the skew on self loops. See Using Timing Analysis with No Clock Skew for Self Loop Paths in the *Common Timing Engine (CTE) User Guide* for more information.

Default: `false`

timing_self_loop_paths_no_skew_max_slack

`timing_self_loop_paths_no_skew_max_slack`

Identifies self-loop paths at the specified threshold. For example:

```
set_global timing_self_loop_paths_no_skew_max_slack 1.0
```

In this case, all timing endpoints with a worst slack less than `1.0` are considered for self loop skew removal. See Using Timing Analysis with No Clock Skew for Self Loop Paths in the *Common Timing Engine (CTE) User Guide* for more information.

Default: `0`

timing_self_loop_paths_no_skew_max_depth

```
timing_self_loop_paths_no_skew_max_depth
```

Identifies self-loop paths at the specified depth. For example:

```
set_global timing_self_loop_paths_no_skew_max_depth 20
```

Lets the timer traverse 20 timing arcs (10 stages of logic) before abandoning the search for a self loop. See Using Timing Analysis with No Clock Skew for Self Loop Paths in the *Common Timing Engine (CTE) User Guide* for more information.

Default: 10

timing_unconstrained_slew_propagation

```
timing_unconstrained_slew_propagation {true | false}
```

When set to `true`, calculates slew on unconstrained portions of a path using the slew propagation method specified by the slew_propagation_mode global. When set to `false`, estimates slew with a less accurate method.

use_global_footprint_for_techlibs

```
use_global_footprint_for_techlibs {on | off}
```

When set to on, maintains a single footprint-name array. Libraries that are loaded in the same session will share the same footprints. This is necessary such that optimization can work on cells that have the same footprints across libraries.

Default: on

write_sdf_force_calculation

```
write_sdf_force_calculation {true | false}
```

Forces the recomputation of delays during `write_sdf` command and writes triplets (`min:typ:max`). Use this option for backward compatibility with 4.0.

Without this option `write_sdf` uses delays already cached in the system and writes only values of the triplets specified by the `pvt_early_path` and `pvt_late_path` globals. This global is equivalent to using the `-force_calculation` option of the `write_sdf` command.

Default: `false`

write_timing_windows_gcf_info

```
write_timing_windows_gcf_info {true | false}
```

When set to `true`, the Timing Window File (TWF) contains the following extra information in addition to the timing windows:

- process, voltage, and temperature parameters
- clock information
- list of Timing Library Format (TLF) files
- timing constraints

If set to `false` none of the extra information is written to the TWF.

Default: `true`

Note: This global only effects the timing windows format for the Celtic crosstalk analysis tool; it does not effect the timing windows format for the Signal Storm crosstalk analysis tool.

12

Temporary Global Variables

Important

This chapter contains descriptions for temporary global variables, which usually begin with an underscore (_). These are intended for use only in the situations described. These globals are often untested, unsupported, and intended only as temporary workarounds. Also, some of these functions are considered high risk, in that they may cause serious problems with your design if not used with extreme caution.

If you are using a temporary global in your flow, would like it documented, or think it should be made permanent, please let us know by using the *CDSDoc Feedback* button or by e-mailing *synthesis_hidden@cadence.com*.

This chapter contains the following sections:

CTPKS

This section describes the hidden globals associated with the clock tree generator in Cadence PKS.

_compact_special_wiring

```
_compact_special_wiring
```

Related Information

set_special_netpin

get_special_netpins

_delete_special_wiring

_display_special_nets

_restore_special_wiring

_ctg_debug_command

```
_ctg_debug_command
```

Related Information

do_build_clock_tree

do_build_physical_tree

get_clock_tree_constraints

get_clock_tree_objects

report_clock_tree

report_clock_tree_violations

set_clock_tree_constraints

_delete_special_wiring

```
_delete_special_wiring
```

Related Information

set_special_netpin

get_special_netpins

_compact_special_wiring

_display_special_nets

_extract_special_wiring

_restore_special_wiring

_display_special_nets

```
_display_special_nets
```

Related Information

set_special_netpin

get_special_netpins

_compact_special_wiring

_delete_special_wiring

_restore_special_wiring

_extract_special_wiring

```
_extract_special_wiring
```

Related Information

set_special_netpin

get_special_netpins

_compact_special_wiring

_delete_special_wiring

_display_special_nets

_restore_special_wiring

_restore_special_wiring

_restore_special_wiring

Related Information

set_special_netpin

get_special_netpins

_compact_special_wiring

_delete_special_wiring

_display_special_nets

_extract_special_wiring

Datapath Synthesis

This section describes the temporary globals associated with Datapath Synthesis in BuildGates Synthesis and Cadence PKS.

_aware_restructuring_effort_level

`_aware_restructuring_effort_level`

Controls the selective restructuring and resynthesis of AmbitWare modules. It is enabled by default in the high effort optimization flow.

Its possible values are:

`none`: Does not restructure and resynthesize AmbitWare modules.

`medium`: Restructures and resynthesizes AmbitWare modules for medium effort optimization.

`high`: Restructures and resynthesizes AmbitWare modules for high effort optimization.

For example, to enable the flow in medium effort:

```
set_global _aware_restructuring_effort_level medium
```

NOTE: This flow provides better runtime than that of the `-restructure_aware` option of the `do_optimize` command. The `do_optimize -restructure_aware` command performs a more elaborate restructuring process than the `_aware_restructuring_effort_level` global.

Design For Test (DFT)

This section describes the hidden globals associated with design for test (DFT) in BuildGates Synthesis and Cadence PKS.

_debug_scan

```
_debug_scan
```

Related Information

check_dft_rules

report_dft_registers

_debug_tdrc

_dump_tdrc

_debug_tdrc

```
_debug_tdrc
```

Related Information

check_dft_rules

report_dft_registers

_debug_scan

_dump_tdrc

_dump_tdrc

```
_dump_tdrc
```

Related Information

check_dft_rules

report_dft_registers

_debug_scan

_debug_tdrc

_dft_allow_ff_pin_renaming

```
_dft_allow_ff_pin_renaming {on|off}
```

When enabled, allows optimization to replace a sequential cell with another sequential cell type without considering the corresponding pin names of the two cells. When disabled, optimization is restricted to a replacement with another cell that has identical pin names as the original cell.

Turn this global `off` after reading a scanDEF file and prior to any optimization runs. This is needed to prevent any of the optimization steps from invalidating the input scanDEF data that has references to sequential instance pin names.

Default: `on`

_dft_resize_ff

```
_dft_resize_ff {on|off}
```

When enabled, optimizes the size (drive strength/area) of the scan flip-flop during its conversion from basic D-flop to its scan equivalent flop given the timing constraints on the scan flop. By default, all basic D-flops which pass the DFT rule checks are remapped to their scan equivalent flops to achieve minimum area.

Default: `off`

HDL

This section describes HDL (Hardware Description Language) temporary globals.

_hdl_enable_graphOpt_across_functaskhier

```
_hdl_enable_graphOpt_across_functaskhier
```

Automatically inlines nodes outside the hierarchies in CDFG. If set to off, automatic inlining does not happen.

_hdl_if_to_case

```
_hdl_if_to_case {true | false}
```

Controls the automatic `if`-to-`case` conversion. The default value is `false`. When the global is set to `true`, the `do_build_generic` command will convert any `if` statement whose conditions are of the form:

```
(<expr> == <const>)
```

into an equivalent `case` statement. The expression, `<expr>,` must be the same in each condition. Such conditions can also be `OR`'ed together.

For example, the following `if` statement:

```
if (x == 1 || x == 2)
        q = 1;
    else if (x == 3)
        q = 2;
```

will be converted into the following equivalent case statement

```
case (x)
      1, 2:
        q = 1;
      3:
        q = 2;
```

```
      default:
    endcase
```

Default: `false`

_hdl_implicit_constant_propagation

```
_hdl_implicit_constant_propagation { true | false }
```

Enables the propagation of constant values within conditional statements.

The following example illustrates a simple `if` statement:

```
if (x == 3)
    y = x + 1;
else
    y = x - 1;
```

With implicit constant propagation enabled, the `do_build_generic` command determines that in the first branch of the `if` statement the value of `x` is `3`. This can simplify the netlist generated for this branch.

Implicit constant propagation also applies to `case` statements:

```
case (x)
    3: y = x + 1;
    default: y = x - 1;
endcase
```

Default: `true`

_hdl_partial_sop_extraction

```
_hdl_partial_sop_extraction { on | off }
```

This global applies to `case` statements that include both constant and non-constant assignments. When the `_hdl_partial_sop_extraction` global is set to `on`, the constant assignments within the `case` statement are identified and extracted into a sum-of-products (SOP) module during `do_build_generic`. The remaining non-constant values are implemented as multiplexers.

Note: A `case` statement must have at least 55% of its assignments be constant assignments for SOP extraction to occur. This threshold can be adjusted using the _hdl_partial_sop_min_percentage global.

Default: `on`

_hdl_partial_sop_min_percentage

```
_hdl_partial_sop_min_percentage { percentage_value }
```

This global applies to `case` statements that include both constant and non-constant assignments. Specifically, it specifies the threshold at which extraction of constant assignments will occur. If the number of constant assignments in a `case` statement is below the specified threshold, they will not be extracted into sum-of-product modules.

Default: 55%

_ucad

```
_ucad
```

_zero_supply_test_setup

```
_zero_supply_test_setup
```

Low Power Synthesis (LPS)

This section describes the hidden globals associated with the low power synthesis functionality in BuildGates Synthesis and Cadence PKS.

Optimization

This section describes temporary globals associated with optimization:

- _opt_reclaim_registers
- _opt_resize_registers_for_area
- _rbo_loc_spiral_levels_for_hold
- _rbo_loc_spiral_levels_for_non_hold
- _rbo_run_multibuf_algorithm
- remove_overlapping_filler_cells_in_opt
- _struct_elim_skip_pos_node
- _topt_enable_pinswap_for_seq_cell

_opt_reclaim_registers

```
_opt_reclaim_registers {true | false}
```

For certain designs reclaiming register area can be very time consuming. This global controls register area reclamation in various phases of optimization. If set to `false`, registers are not down-sized in either the initial area reclamation phase or in any subsequent area reclamations to improve placement bin capacity.

Note: This global affects the area of the design and may have an impact on final quality of results as well.

Default: `true`

_opt_resize_registers_for_area

```
_opt_resize_registers_for_area {true | false}
```

Same as `_opt_reclaim_registers`.

_rbo_loc_spiral_levels_for_hold

```
-rbo_loc_spiral_levels_for_hold
```

Specifies the number of spiral levels the optimization engine can search for a location to insert a buffer. Increasing the value enlarges the search area for a placement bin with sufficient space to hold a new buffer. Using this global can result in longer run time.

This global is useful when optimization gets stuck while fixing hold violations, and you believe that one part of the design may be getting full. In such a situation, you can increase the value of the global to a value greater than 6 (the default).

Default: 6

_rbo_loc_spiral_levels_for_non_hold

```
_rbo_loc_spiral_levels_for_non_hold
```

Specifies the number of spiral levels the optimization engine can search for a location to insert a buffer. Increasing the value enlarges the search area for a placement bin with sufficient space to hold a new buffer. Using this global can result in longer run time.

This global is useful when optimization gets stuck while fixing design rule or setup violations, and you believe that one part of the design may be getting full. In such a situation, you can increase the value of the global to a value greater than 3 (the default).

Default: 3

_rbo_run_multibuf_algorithm

```
_rbo_run_multibuf_algorithm {on | off}
```

When set to `on`, instructs PKS to try a multi-buffer algorithm during design rule violation fixing. This is the same algorithm used during large-fanout net fixing. If the netlist contains many long or high-fanout nets, the effect should be to reduce running time during DRV fixing.

Default = `off`

remove_overlapping_filler_cells_in_opt

```
remove_overlapping_filler_cells_in_opt {true | false}
```

By default, all filler cells are removed from the netlist prior to optimization and must be reinserted as a separate step after optimization completes.

If PKS is optimizing a placed design containing filler cells, and this global is set to `true`, PKS optimizes the design, and then removes only those filler cells that are overlapping new or modified instances. All non-violating filler cells from the original netlist and their placement are retained.

Default: `false`

_struct_elim_skip_pos_node

```
_struct_elim_skip_pos_node {on |off}
```

Controls node collapsing. When this global is set to `on`, nodes with AND logic followed by OR logic are not collapsed. This type of node occurs mostly in POS (product of sum) logic networks. Without this global, POS trees can sometimes be difficult to optimize because collapsing such nodes can create larger, unoptimized nodes. This global helps retain the original, smaller node structures.

Default: `off`

_topt_enable_pinswap_for_seq_cell

```
_topt_enable_pinswap_for_seq_cell
```

Provides a workaround for cases where active-low reset (or active-high set) is connected to data and vice-versa.

PKS

This section describes temporary globals associated with Cadence PKS:

_pb_td_eco

```
_pb_td_eco {0 | 2}
```

Attempts local placement moves along the critical path to improve timing, if set to 2 prior to running your optimization command (`do_optimize` or `do_optimize -ipo`). By default, local placement moves are not attempted. Set this global to 2 if you have a timing critical flow, especially where large slack jumps occur during the placement legalization step.

Default: `0`

Note: This global and the underlying transforms will likely be made part of the default flow in the future.

_pks_def_net_route_from_rt_graph

```
_pks_def_net_route_from_rt_graph {on | off}
```

When set to `on`, causes the `write_def` route writer to revert to the previous behavior which is to *not* write out route extensions. Set this global before issuing the `write_def` command.

Default: `off`

_pks_handle_def_groups

```
_pks_handle_def_groups {0 | 1}
```

When set to `0`, subsequent `read_def` commands will not read the `GROUP` section.

Default: `1`

_pks_ignore_gr_term_pessimism

```
_pks_ignore_gr_term_pessimism {on | off}
```

By default, for a globally routed net, the RC values consist of two parts: one is for the edges in the route graph with an actual layer assigned. RCs from such edges are computed accurately, based on LEF layer RC and edge length. The other part is for edges connecting a pin and the center of the gcell containing the pin. When this global is set to `on`, RC from the second part is ignored.

Default: `off`

_pks_setup_wireload

```
_pks_setup_wireload
```

Forces optimization to use an internally-generated wireload model from PKS instead of the default library wire load model.

Related Information

`set_wire_load`

`reset_wire_load`

`set_wire_load_selection_table`

`reset_wire_load_selection_table`

`set_port_wire_load`

`set_wire_load_mode`

_pks_opt_re_place

Controls the re-placement transform during PKS optimization. See enable_re_placement on page 129 for more information.

`_pks_opt_re_place` can take three values:

Value	Description
0	Re-placement is disabled.
1	Re-placement is performed after many of the other optimization transforms (default).
2	Re-placement is performed before most other optimization transforms (With this setting, re-placement is invoked much more frequently than with the 1 setting).

`_pks_opt_re_place` is a supplementary global to enable_re_placement:

- If `enable_re_placement` is `true`, and `_pks_opt_re_place` is `0`, re-placement operates as if `_pks_opt_re_place` were set to `1`.
- If `enable_re_placement` is `true`, and `_pks_opt_re_place` is `1` or `2`, then the behavior is determined by `_pks_opt_re_place`.
- If `enable_re_placement` is `false`, then re-placement behavior is determined entirely by `_pks_opt_re_place`.

The recommended usage for re-placement is:

```
set_global enable_re_placement true
```

An alternative usage is obtained by:

```
set_global _pks_opt_re_place 2
```

This usage may be beneficial on some designs, but keep in mind that it may move a large number of instances, which in turn could make timing convergence more difficult. The recommended usage will move fewer instances.

Default: 1

_rbo_auto_resize_for_reducenetc

```
_rbo_auto_resize_for_reducenetc {on | off}
```

If one of the following commands fails to insert a buffer, this global instructs PKS to try to upsize the driver in order to obtain the same slew at the driver pin that would have been achieved by cutting net load by the fraction specified by the `reduceNetC` or `receiverReduceNetC -value` option:

- `do_xform_run_repair_file reduceNetC`
- `do_xform_run_repair_file receiverReduceNetC`.

_rbo_no_new_ports

```
_rbo_no_new_ports { on | off }
```

Prevents the insertion of any buffers that would result in new ports being added to any modules in the netlist during post-placement buffer insertion, if set to `on`.

Caution

Turning on this global may result in worse QOR, such as setup slack and design rule violations

Default: `off`

Timing

This section describes temporary globals associated with Cadence timing analysis.

_dcalc_new_rd_calculator

```
_dcalc_new_rd_calculator { true | false }
```

Controls the calculation of the driver resistance used by the parasitic delay calculator. Previously, the algorithm was inaccurate for widely spaced measured slew thresholds and for 10/90 percent thresholds. The new algorithm corrects this problem. Set this global to `false` to restore the previous behavior.

Default: `true`

_timing_spf_zero_resistance_threshold

```
_timing_spf_zero_resistance_threshold
```

Sets the minimum resistance value. All resistance value smaller than the specified value will be regarded as zero.

_write_timing_windows_clock_arrival

```
_write_timing_windows_clock_arrival { true | false }
```

Set to `false` by default, so that a clock arrival time assertion set using `set_clock_arrival_time` command will not be listed in the timing windows file. Only the `set_clock_root` assertion is listed.

_write_timing_windows_fast_mode

```
_write_timing_windows_fast_mode { true | false }
```

Set to `true` by default, so that the `write_timing_windows` command will not create hash tables to sort out the timing windows per clock phase. You will see approximately 20 percent runtime improvement using the original test case.

When the global is set to `false`, the `write_timing_windows` command behaves in the old mode where it sorts the timing windows per clock phase and takes longer time.

_write_timing_window_slack_info

```
_write_timing_window_slack_info { true | false }
```

Set to `true` by default because CeltIC™ uses the slack information for better accuracy. However, setting this global to `false` improves runtime.

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

W