Formality®
Variables and Attributes

Version T-2022.03-SP4, September 2022

SYNOPSYS®

Copyright Notice and Proprietary Information

Destination Control Statement

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

Disclaimer

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

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at
http://www.synopsys.com/company/legal/trademarks-brands.html.
All other product or company names may be trademarks of their respective owners.

Free and Open-Source Software Licensing Notices

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

Third-Party Links

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

www.synopsys.com

Contents

alternate_strategy_job_env

Specifies the compute farm to be used for running alternate strategies.

TYPE

string

DEFAULT

"GRD"

ENABLED SHELL MODES

all

DESCRIPTION

Use this variable to select the compute farm for running alternate strategies.

The default value of this variable is "GRD".

To change the value of this variable, enter **set alternate_strategy_job_env** *value*, where *value* is "GRD", "LSF" or "LOCAL".

EXAMPLES

The following example changes the value of **alternate_strategy_job_env** to "LOCAL".

```
fm_shell> set alternate_strategy_job_env "LOCAL"
```

SEE ALSO

alternate_strategy_monitor_env(3)
alternate_strategy_job_options(3)
alternate_strategy_monitor_options(3)

alternate_strategy_job_options

Specifies the options to be passed to the compute farm while running alternate strategies.

TYPE

string

DEFAULT

""

ENABLED SHELL MODES

all

DESCRIPTION

Use this variable to specify the options to be passed to the compute farm while running alternate strategies.

The default value of this variable is "".

To change the value of this variable, enter **set alternate_strategy_job_env** *value*, where *value* is a function of the compute farm being used.

This variable has no impact if the alternate strategies are being run on the local machine.

EXAMPLES

The following example changes the value of **alternate_strategy_job_options** to reserve a GRD machine with 8GB of memory.

```
fm_shell> set alternate_strategy_job_options {-l mem_free=8G}
```

SEE ALSO

alternate_strategy_job_env(3)
alternate_strategy_monitor_env(3)
alternate_strategy_monitor_options(3)

alternate_strategy_monitor_env

Specifies the compute farm to be used for running monitor process for alternate strategies.

TYPE

string

DEFAULT

""

ENABLED SHELL MODES

all

DESCRIPTION

Use this variable to select the compute farm for running monitor process for alternate strategies.

The default value of this variable is "".

To change the value of this variable, enter **set alternate_strategy_monitor_env** *value*, where *value* is "" or "LOCAL".

If this variable is set to "", the monitor process is run on the same farm as the alternate strategies. If it is set to "LOCAL", the monitor process is run on the local machine independent of the compute resources specified for the alternate strategies.

EXAMPLES

The following example changes the value of **alternate_strategy_monitor_env** to "LOCAL".

```
fm_shell> set alternate_strategy_monitor_env "LOCAL"
```

SEE ALSO

alternate_strategy_job_env(3)
alternate_strategy_job_options(3)
alternate_strategy_monitor_options(3)

alternate_strategy_monitor_options

Specifies the options to be passed to the compute farm while running the monitor process for alternate strategies.

TYPE

string

DEFAULT

""

ENABLED SHELL MODES

all

DESCRIPTION

Use this variable to specify the options to be passed to the compute farm while running the monitor process for alternate strategies.

The default value of this variable is "".

To change the value of this variable, enter **set alternate_strategy_monitor_options** *value*, where *value* is a function of the compute farm being used.

This variable has no impact if the monitor process for alternate strategies is run on the local machine.

EXAMPLES

The following example changes the value of **alternate_strategy_monitor_options** to reserve a GRD machine with 1GB of memory.

```
fm_shell> set alternate_strategy_monitor_options {-l mem_free=1G}
```

SEE ALSO

alternate_strategy_job_env(3)
alternate_strategy_monitor_env(3)
alternate_strategy_job_options(3)

architecture_selection_precedence

Determines the precedence of TCL variables and commands used to define architectures for multiplier instances. This variable can only be changed during setup mode.

TYPE

string

DEFAULT

"set_architecture svf fm_pragma hdlin_multiplier_architecture"

DESCRIPTION

When multiplier generation is enabled (see **enable_multiplier_generation**), use this variable to define the precedence of the various architecture selection methods available. With the variable set to its default value, for a given multiplier instance, Formality will first look for a **set_architecture** command that defines an architecture for that instance. If none is found, Formality will look for an architecture specified with the source pragma and finally for the architecture specified with **hdlin_multiplier_architecture**. If no architecture is found using any of these methods, Formality will attempt to choose the architecture Design Compiler was likely to have picked for each multiplier instance.

Before issuing a read command, change the value of this variable as follows:

set architecture_selection_precedence *value*

where *value* is a space seperated list of the following identifiers: set_architecture, fm_pragma and hdlin_multiplier_architecture. The meaning of these identifiers is:

* set_architecture - Use the set_architecture command to select a multiplier architecture.

* fm_pragma - Use the value of the Formality pragma in the HDL code to select a multiplier architecture.

* hdlin_multiplier_architecture - Use the value of the TCL variable hdlin_multiplier_architecture at the time a design file was read to select a multiplier architecture.

This variable will not have any affect unless **enable_multiplier_generation** is set to true.

SEE ALSO

report_architecture(2)
set_architecture(2)
enable_multiplier_generation(3)
hdlin_multiplier_architecture(3)
dw_foundation_threshold(3)

bus_dimension_separator_style

Sets the VHDL/Verilog read option that controls the bus dimension separator style used with multi-dimensional buses.

TYPE

string

DEFAULT

"]["

DESCRIPTION

Use this variable to set bus dimension separators. You can supply any characters as the bus dimension separators. You must enclose them in braces { } or quotes " ".

To change the value of this variable, enter **set bus_dimension_separator_style** "*value*".

Once set, this option remains in effect for the entire verification session.

In this example, the designs use the following naming scheme for multi-dimensional buses:

```
BUSB_1_1
BUSA_1_2
BUSA_1_3
```

In this case, you can define the bus separator style as the underscore character "_" by using the following command.

```
fm_shell> set bus_dimension_separator_style "_"
```

SEE ALSO

bus_naming_style(3)

bus_extraction_style

Sets the VHDL/Verilog read option that controls the bus extraction style.

TYPE

string

DEFAULT

"%s[%d:%d]"

DESCRIPTION

Use this variable to set the VHDL/Verilog read option that controls the bus extraction style. Once set, this option remains in effect for all future read operations during this session. You must enclose the value in braces {} or quotes "".

To change the value of this variable, enter **set bus_extraction_style** "*value*".

SEE ALSO

bus_naming_style(3)

bus_multiple_separator_style

Separator style used to decode the name of a multibit cell that implements bits that do not form a range.

TYPE

string

DEFAULT

","

DESCRIPTION

The variable is used during the processing of multibit guidance to decode the name of a multibit cell that implements bits that do not form a range.

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

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

Decoding of 6-bit wide multibit cell name q[0:2,5:7] indicates that the cells q[0], q[1], q[2], q[5], q[6] and q[7] are packed together. Also, decoding of 4-bit wide multibit cell name q[0,2,4,6] indicates that the cells q[0], q[2], q[4] and q[6] are packed together.

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

To change the value of this variable, enter **set bus_multiple_separator_style** "*value*".

SEE ALSO

bus_naming_style(3)
bus_range_separator_style(3)

bus_naming_style

Sets the VHDL/Verilog read option that controls the bus naming style.

TYPE

string

DEFAULT

"%s[%d]"

DESCRIPTION

Use this variable to set the bus naming style. When supplying a bus name style, enclose it in braces {} or quotes "" so the string includes the ordered character pairs *%s* and *%d* exactly as they appear. Within the quoted string, you can use any character or string of characters to replace the opening and closing brackets used in the default naming scheme. For example, suppose your designs use the following naming scheme for buses:

```
BUSA_1
BUSA_2
BUSA_3
```

In such a case, the following command ensures Formality interprets bus names correctly.

```
fm_shell> set bus_naming_style {%s_%d}
```

To change the value of this variable, enter **set bus_naming_style** "*value*".

Once set, this option remains in effect for the entire verification session.

SEE ALSO

bus_dimension_separator_style(3)
bus_extraction_style(3)

bus_range_separator_style

Sets the EDIF/VHDL/Verilog read option that controls the bus range separator style.

TYPE

string

DEFAULT

":"

DESCRIPTION

Use this variable when supplying a bus range separator style. You can supply any characters as the bus range separator. You must enclose them in braces {} or quotes "". The bus range separator is used only to identify banked register cells in a gate level netlist.

To change the value of this variable, enter **set bus_range_separator_style** "*value*".

Once set, this option remains in effect for the entire verification session.

For example, suppose your gate level netlist instantiates banked cells that use the following naming scheme :

```
MEM[3to0]
MEM[7to4]
...
```

In this case, you can define the bus range style as the character string "to" by using the following command.

```
fm_shell> set bus_range_separator_style "to"
```

SEE ALSO

bus_naming_style(3)
bus_multiple_separator_style(3)

collection_result_display_limit

Sets the maximum number of objects that will be displayed for commands producing collection results.

TYPE

integer

DEFAULT

100

DESCRIPTION

Sets the maximum number of objects that will be displayed for any command that results in a collection. The default is *100*.

When a command, such as the **add_to_collection** command, is issued at the command prompt, the result is implicitly queried, as though the **query_objects** command was called. You can limit the number of objects displayed by setting the **collection_result_display_limit** variable to an appropriate integer. A value of *-1* displays all objects.

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

printvar collection_result_display_limit

SEE ALSO

collections(2)
query_objects(2)

confirmed_SVA

Confirms the interpretation of SystemVerilog assertions as constraints

TYPE

Boolean

DEFAULT

DESCRIPTION

The **confirmed_SVA()** Tcl variable confirms that a labeled assertion should be converted to a constraint. The array *key* is a pathname starting with the module name that identifies the assertion.

To confirm the assertions in the following module:

```
module test(sel, o, clk);
input [1:0] sel;
input clk;
output reg o;

always@(posedge clk)
  begin : clocked
    reg [1:0] sel1;
    sel1 <= sel;
    sel_rl: assert final (sel1 >= 1);
    sel_ru: assert final (sel1 < 2);
    priority case(sel1)
       3'b01:   o <= 1'b1;
       3'b10:   o <= 1'b0;
       default: o <= 1'b1;
    endcase
  end
endmodule
```

You would set the following **confirmed_SVA** array values:

```
set confirmed_SVA(test.clocked.sel_rl) true
set confirmed_SVA(test.clocked.sel_ru) true
```

SEE ALSO

hdlin_enable_assertions(3)

diagnosis_enable_error_isolation

Enables the precise diagnosis engine of Formality.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Enables the precise diagnosis engine of Formality.

Use this boolean variable to enable/disable the precise diagnosis engine. The **diagnose**, **report_error_candidates** commands and the "Error Candidates" tab in the main GUI will be affected by this variable. With this engine, no automatic diagnosis is performed when you invoke a cone view. The recommended flow is to manually perform "diagnose" first. If this did not return any error candidates, try diagnosing a related group of failing points.

SEE ALSO

diagnose(2)
report_error_candidates(2)

diagnosis_pattern_limit

Specifies the maximum number of failing patterns that Formality uses during diagnosis.

TYPE

integer

DEFAULT

256

DESCRIPTION

Use this variable to set a limit to failing patterns used during diagnosis.

Formality generates many failing patterns during a failed verification. By default, Formality uses a maximum of 256 failing patterns during diagnosis.

To change the value of this variable, enter **set diagnosis_pattern_limit** *value*, where *value* is an integer.

SEE ALSO

diagnose(2)

dpx_auto_session_interval

Specifies a wall-clock time interval for which sessions are automatically saved during distributed computing.

TYPE

string or integer

DEFAULT

"0:0:0"

DESCRIPTION

Use this variable along with the *verification_auto_session* variable to specify the time interval that must occur before the Formality tool automatically saves a session file during distributed computing. This variable takes effect only when the *verification_auto_session* variable is set to 'on', 'always' or 'verify'. Only the last automatically saved session file is retained.

The default of the *dpx_auto_session_interval* variable is 0 hours. A value of 0 disables automatic saving of session files during distributed computing. You can use positive integers for hours or the hours:minutes:seconds format.

To change the value of this variable, enter the **set dpx_auto_session_interval** *value*, where *value* is a positive integer for hours or is in the hours:minutes:seconds format.

For example, to indicate a 30-minute interval, specify 0:30:00. If you specify 30, Formality interprets it as 30:0:0; that is, a 30-hour interval.

SEE ALSO

set_dpx_options(2)
verify(2)
verification_auto_session(3)
dpx_enable_checkpoint_verification(3)

dpx_enable_checkpoint_verification

Enables DPX verification for checkpoints.

TYPE

boolean

DEFAULT

"true"

ENABLED SHELL MODES

All

DESCRIPTION

Checkpoint verifications are performed during the preverification (SVF processing) stage. By default, DPX is deployed for checkpoint verifications.

In a DPX run, you can set this variable to false to disallow DPX to be used for checkpoint verifications.

EXAMPLES

The following example turns DPX off for checkpoint verifications:

```
fm_shell (setup)> set dpx_enable_checkpoint_verification false
fm_shell (setup)> set_dpx_options -protocol SH -submit_command sh -max_workers 1 -max_cores 8
```

SEE ALSO

set_dpx_options(2)
remove_dpx_options(2)
report_dpx_options(2)
start_dpx_workers(2)
stop_dpx_workers(2)
get_dpx_workers(2)

dpx_keep_workers_alive(3)
dpx_verification_strategies(3)
dpx_worker_acquisition_timeout(3)

dpx_ignored_strategies

Specifies certain verification strategies to not run during distributed verification.

TYPE

string

DEFAULT

Empty

ENABLED SHELL MODES

All

DESCRIPTION

A distributed verification task is a combination of a partition to be verified and a verification strategy to be run. A given partition may be verified using many different verification strategies, each one part of a different task. When any one task completely verifies a partition, any other task running that partition with a different strategy is terminated so that the worker slot can be made available for other tasks.

By default, which is when the dpx_ignored_strategies variable is empty, Formality will use a number of strategies and combinations of strategies that are available for distributed verification tasks.

If set, the dpx_ignored_strategies variable controls when certain strategies and combo strategies should not be used for distributed processing. If it is known that certain strategies are particularly ineffective for verifying designs, those strategies should not appear in the list of strategies to be run by DPX.

The format of the strategy list provided for the dpx_verification_strategies variable is as follows:

```
{str1 str2 str3 {str1 str5} str4 {str2 str5 str3}}
```

For this format, strN denotes a specific strategy, that is, s1, o2, and so on. The entire string is enclosed within curly braces, and combination strategies are embedded within the list in curly braces. The special strategy "none" is the default strategy used by Formality in non-DPX runs.

An example strategy list is as follows:

```
set dpx_ignored_strategies {none l1 {s1 s4} q1 {s3 s8}}
```

EXAMPLES

The following example defines the strategies to be ignored during distributed verification:

```
fm_shell (setup)> set dpx_ignored_strategies {none l1 {s1 s4} q1 {s3 s8}}
none l1 {s1 s4} q1 {s3 s8}
```

SEE ALSO

dpx_overall_strategy(2)
dpx_verification_strategies(2)
verification_alternate_strategies(2)
set_dpx_options(2)
remove_dpx_options(2)
report_dpx_options(2)

dpx_keep_workers_alive

Controls whether DPX workers remain available when not in use.

TYPE

boolean

DEFAULT

"true"

ENABLED SHELL MODES

All

DESCRIPTION

By default, DPX does not release workers until the end of the existing session, or until they are explicitly released with the **stop_dpx_workers** command.

Setting this variable to false instructs DPX to release workers after each distributed stage.

Distributed processing can be invoked at multiple points in the session, that is, during the preverification (SVF processing) stage, and then again during the full verification stage.

The default DPX behavior is to not release workers between processing stages. Some compute farm infrastructures might have a large delay between the request for compute resources (workers) and their availability. This can lead to DPX inefficiencies because workers that are not available cannot be used for distributed processing. If your compute farm experiences long wait times before resources are provisioned, you might want to avoid releasing workers between distributed processing stages.

On the other hand, between distributed processing stages there could be long periods of inactivity for the workers. By setting the variable to false, DPX will release the workers after each stage freeing up compute resources.

EXAMPLES

The following example instructs DPX to release workers between processing stages:

```
fm_shell (setup)> set dpx_keep_workers_alive false
false
```

SEE ALSO

set_dpx_options(2)
remove_dpx_options(2)
report_dpx_options(2)
start_dpx_workers(2)
stop_dpx_workers(2)
get_dpx_workers(2)
dpx_enable_checkpoint_verification(3)
dpx_verification_strategies(3)
dpx_worker_acquisition_timeout(3)

dpx_verification_strategies

Defines the verification strategies deployed during distributed verification.

TYPE

string

DEFAULT

Empty

ENABLED SHELL MODES

All

DESCRIPTION

A distributed verification task is a combination of a partition to be verified and a verification strategy to be run. A given partition may be verified using many different verification strategies, each one part of a different task. When any one task completely verifies a partition, any other task running that partition with a different strategy is terminated so that the worker slot can be made available for other tasks.

By default, which is when the dpx_verification_strategies variable is empty, Formality will use a number of strategies and combinations of strategies that are available for distributed verification tasks.

If set, the dpx_verification_strategies variable controls which strategies and combo strategies are used for distributed processing. It also controls the order in which those strategies are deployed. If it is known that certain strategies are particularly effective for verifying your designs, those strategies should appear earlier in the list of strategies.

The strategies specified by dpx_verification_strategies are attempted first. If, after attempting those strategies, there are still unverified points left, Formality will attempt the verification with other strategies.

The format of the strategy list provided for the dpx_verification_strategies variable is as follows:

```
{str1 str2 str3 {str1 str5} str4 {str2 str5 str3}}
```

For this format, strN denotes a specific strategy, that is, s1, q2, and so on. The entire string is enclosed within curly braces, and combination strategies are embedded within the list in curly braces. The special strategy "none" is the default strategy used for serial verification.

An example strategy list is as follows:

```
set dpx_verification_strategies {none s4 {s1 s4} s3 {s3 s8}}
```

EXAMPLES

The following example defines the verification strategies deployed during distributed verification:

```
fm_shell (setup)> set dpx_verification_strategies {none s4 {s1 s4} s3 {s3 s8}}
none s4 {s1 s4} s3 {s3 s8}
```

SEE ALSO

set_dpx_options(2)
remove_dpx_options(2)
report_dpx_options(2)
start_dpx_workers(2)
stop_dpx_workers(2)
get_dpx_workers(2)
dpx_enable_checkpoint_verification(3)
dpx_keep_workers_alive(3)
dpx_worker_acquisition_timeout(3)

dpx_worker_acquisition_timeout

Specifies a wall-clock time limit to acquire first worker for distributed computing.

TYPE

string or integer

DEFAULT

"0"

DESCRIPTION

Use this variable to specify the maximum wall-clock time allowed for the **preverify**, **match**, and **verify** commands to wait for the first worker to be acquired and ready to accept distributed tasks.

The **preverify**, **match**, and **verify** commands either run to completion or stop when the value specified by **dpx_worker_acquisition_timeout** (default is unlimited) is reached without getting any worker. If the time limit specified is reached, the tool will interrupt the current state of verification.

You must enter positive integers for hours and minutes. For no time restriction specify *none* or *0* or *0:0:0*.

To change the value of this variable, enter **set dpx_worker_acquisition_timeout** *value*, where *value* is an integer or in hours, minutes, and/or seconds using the following format: hours:minutes:seconds.

For example, to indicate a 60 second time limit, specify 0:0:60. If you specify 60, Formality interprets it as 60:0:0; that is, a 60 hour time limit.

SEE ALSO

preverify(2)
match(2)
verify(2)
set_dpx_options(2)
remove_dpx_options(2)
report_dpx_options(2)
start_dpx_workers(2)
stop_dpx_workers(2)
get_dpx_workers(2)
dpx_enable_checkpoint_verification(3)
dpx_keep_workers_alive(3)
dpx_verification_strategies(3)

dw_foundation_threshold

Defines the input width where Formality will switch from generating nbw to wall architectures for multiplier instances. Only available during setup mode.

TYPE

integer

DEFAULT

52

DESCRIPTION

When multiplier generation is enabled (see **enable_multiplier_generation**), this variable controls the threshold at which Formality switches from generating nbw to wall architectures for multiplier instances. When **hdlin_multiplier_architecture** is set to **dw_foundation**, Formality uses **dw_foundation_threshold** to try and select the same architecture Design Compiler was likely to have selected for that architecture. In general, DC chooses the nbw architecture when the combined width of the multiplier inputs is "small" and it chooses wall architectures for multipliers with larger input widths. Thus, when the combined input widths of a given multiplier instance are less than or equal to **dw_foundation_threshold**, Formality will generate an nbw architecture for that instance. When the combined widths are greater than **dw_foundation_threshold**, Formality will generate a wall architecture.

During setup mode, change the value of this variable as follows:

```
set dw_foundation_threshold value
```

where *value* must be an integer greater than or equal to zero.

This variable will not have any affect unless **enable_multiplier_generation** is set to true.

SEE ALSO

enable_multiplier_generation(3)
hdlin_multiplier_architecture(3)
architecture_selection_precedence(3)

eco_impl

Indicates the current ECO implementation design.

TYPE

string

DEFAULT

DESCRIPTION

This variable is a read-only variable set by the application to indicate the current ECO implementation design specified with the set_eco_implementation command.

SEE ALSO

eco_ref(3)
orig_ref(3)
orig_impl(3)

eco_ref

Indicates the current ECO reference design.

TYPE

string

DEFAULT

DESCRIPTION

This variable is a read-only variable set by the application to indicate the current ECO reference design specified with the set_eco_reference command.

SEE ALSO

eco_impl(3)
orig_ref(3)
orig_impl(3)

enable_multiplier_generation

Enables Formality to generate multiplier architectures based on user directives. This variable can only be changed during setup mode.

TYPE

boolean

DEFAULT

true

DESCRIPTION

Use this variable to enable Formality to generate multiplier architectures for all multiplier instances in the reference design. Unless this variable is turned on, the following commands and TCL variables will have no affect: **set_architecture**, **report_architecture**, **hdlin_multiplier_architecture**, **architecture_selection_precedence** and **dw_foundation_threshold**.

To change the value of this variable, enter:

set enable_multiplier_generation *value*

where *value* is true or false.

SEE ALSO

report_architecture(2)
set_architecture(2)
hdlin_multiplier_architecture(3)
architecture_selection_precedence(3)
dw_foundation_threshold(3)

equivalent_nets_timeout_limit

Specifies a wall time limit for the **find_equivalent_nets** command.

TYPE

string or integer

DEFAULT

"00:15:00"

DESCRIPTION

This variable specifies the maximum time used to search for equivalent nets using the **find_equivalent_nets** command.

To limit the time used to search for equivalent nets, set this variable to a maximum wall-clock time, which is applied to subsequent **find_equivalent_nets** calls. Formality stops the search for equivalent nets when it reaches the time limit.

You must enter positive integers for hours, minutes, and seconds. Use "None" or "0:0:0" to place no time limit on **find_equivalent_nets**, in which case, the command will run until all equivalences for the input nets have been found.

To change the timeout limit, set this variable using the following format: hours:minutes:seconds.

EXAMPLES

To indicate a 60 second time limit, specify 0:0:60.

```
fm_shell (setup)> set equivalent_nets_timeout_limit 0:0:60
```

SEE ALSO

find_equivalent_nets(2)

fm_work_path

Specifies the name of the Formality work directory.

TYPE

string

DEFAULT

The directory from where Formality is invoked.

DESCRIPTION

This read-only variable contains the location of the Formality work directory.

To specify a non-default location, use the "-work_path" command-line option when starting Formality.

SEE ALSO

fm_shell(1)

formality_log_name

Specifies the name of the formality log file. Read-only.

TYPE

string

DEFAULT

"formality.log"

DESCRIPTION

This read-only variable indicates the file name of the Formality log file.

To specify a custom log file name, use the "-name_suffix" command line switch when starting Formality. Formality will append the string given with the "-name_suffix" switch to the log file name.

SEE ALSO

fm_shell(1)

golden_upf_report_missing_objects

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

When this variable is set to *true*, and **load_upf -strict_check false** is processed, Formality reports informational messages about unresolved name references in the UPF. By default, with **load_upf -strict_check false**, unresolved name references are silently ignored.

SEE ALSO

load_upf(2)

golden_upf_version

TYPE

string

DEFAULT

<latest supported version of Golden UPF>

DESCRIPTION

This read-only variable reports the latest version of the Golden UPF flow supported by the Formality tool.

EXAMPLE

```
fm_shell (setup)> get_app_var golden_upf_version
v1.0
fm_shell (setup)>
```

SEE ALSO

load_upf(2)

gui_report_length_limit

Specifies a GUI report size limit.

TYPE

integer

DEFAULT

2000000

DESCRIPTION

Use this variable to control the maximum number of compare points to be displayed in each of the GUI's compare point reports (e.g. Failing Points, Passing Points, etc.). This can be useful to control the amount of data that must be uploaded by the GUI during initialization. A value of 0 will set no limit, allowing all compare point data to be uploaded.

To change the value of this variable, enter **set gui_report_length_limit** *value*, where *value* is a non-negative integer.

gui_transcript_line_length

Specifies a length limit for lines in the GUI transcript.

TYPE

integer

DEFAULT

256

DESCRIPTION

Use this variable to control truncation of lines displayed in the GUI transcript window.

To change the value of this variable, enter **set fBgui_transcript_line_length** *value*, where *value* is a non-negative integer.

guide_checkpoint_auto_session

Specifies if a session file will be generated automatically.

TYPE

String

DEFAULT

"off"

DESCRIPTION

This variable controls automatic saving of a session file at various points during and after guide_checkpoint pre-verification.

Specify one of the following values:

- *off* - No session file is generated.
- *timeout* - A session file is generated after verification timeout.
- *on* - A session file is generated after verification timeout, or also after verification terminates for any reason unsuccessfully after a threshold determined by the variable *verification_auto_session_threshold*.
- *always* - A session file is generated when verification finishes or times-out regardless of the threshold in variable *guide_checkpoint_auto_session_threshold*.
- *verify* - A session file is generated as when the value is *on* and also after each effort level of the **verify** command. Only the last session file is saved.

A session file with the name **checkpoint_<type>.fss** will be automatically created in the directory where all other auto generated files reside (log files etc.). If a file of that name already exists when verification starts, it will be named by inserting the smallest integer value (starting with one) needed to make it unique (example **formality1_timeout_session.fss**).

EXAMPLE

```
fm_shell (setup)> printvar verification_auto_session
verification_auto_session = "on"
fm_shell (setup)> set verification_auto_session match
match
```

SEE ALSO

guide_checkpoint_auto_session_threshold(3)
guide_checkpoint_timeout_limit(3)

guide_checkpoint_auto_session_threshold

Specifies a wall-clock time after which sessions are automatically saved.

TYPE

string or integer

DEFAULT

"0:0:0"

DESCRIPTION

Use this variable to specify the period of time that must occur before the Formality tool automatically saves a session file for a guide_checkpoint pre-verification. When enabled, formality will write a session in "<WORK>/guide_checkpoint_incremental_session/<checkpoint_type>_<operation_id>_<design_name>.fss". Each time the time interval is reached the previous session will be overwritten if it exists.

The default of the *guide_checkpoint_auto_session_threshold* variable is 0 (disabled). Enter positive integers for hours or use the hours:minutes:seconds format.

To change the value of this variable, enter **set guide_checkpoint_auto_session_threshold** *value*, where *value* is a positive integer or is in the hours:minutes:seconds format.

For example, to indicate a 30 minute threshold, specify 0:30:00. If you specify 30, Formality interprets it as 30:0:0; that is, a 30 hour threshold.

SEE ALSO

verification_auto_session(3)

guide_checkpoint_timeout_limit

Specifies a wall-clock time limit for guide_checkpoint pre-verification.

TYPE

string or integer

DEFAULT

"00:0:0"

DESCRIPTION

Use this variable to specify maximum wall-clock time allowed for **guide_checkpoint** pre-verifications.

The **guide_checkpoint** pre-verification will timeout after specified time is reached. If the time limit specified is reached, the tool will interrupt the current state of verification. svf processing will continue after the timeout

You must enter positive integers for hours and minutes. To perform verification until design equivalence is either proven or disproven, specify *none*.

To change the value of this variable, enter **set guide_checkpoint_timeout_limit** *value*, where *value* is an integer or in hours, minutes, and/or seconds using the following format: hours:minutes:seconds.

For example, to indicate a 60 second time limit, specify 0:0:60. If you specify 60, Formality interprets it as 60:0:0; that is, a 60 hour time limit.

SEE ALSO

preverify(2)
match(2)
verify(2)

hdl_naming_threshold

Determines the maximum character length that a parameter can have to be included in the design name.

TYPE

Integer

DEFAULT

-1

DESCRIPTION

Determines the maximum character length that a parameter can have in order to be included in the design name. Normally, design names include the values of the design's parameters or generics. Parameters with character length greater than the current value of this variable are omitted from the design name. A negative value for this variable disables the name threshold check.

SEE ALSO

template_naming_style(3)
template_parameter_style(3)
template_separator_style(3)

hdlin_allow_partial_pg_netlist

Allows Formality to use a partial power ground aware netlist when UPF is not applied.

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

When UPF is applied, Formality assumes all technology library cells in the design to be power and ground aware. If UPF is applied then setting this variable will have no effect. When UPF is not applied, setting this variable to "true" will cause Formality to allow use of a partial power ground aware netlist. Some of the technology library cells can use the power aware version of the cell and other cells in the netlist can use the non-power aware version of the cell.

To change the value of this variable, use the following command: **set hdlin_allow_partial_pg_netlist** "*value*" where *value* is "true" or "false".

Setting this variable to "false" when a partial PG design is read with no load_upf would cause an error at the beginning of match since the reference design having missing PG pin connections with no UPF loaded can cause incorrect verification results.

hdlin_analyze_verbose_mode

Queries information about preprocessing of Verilog and Sys- temVerilog conditional compilation compiler directives and macro expansions.

TYPE

integer

DEFAULT

0

DESCRIPTION

The hdlin_analyze_verbose_mode variable lets you query information about preprocessing of the Verilog or SystemVerilog RTL, including macro expansions and conditional compilation compiler directives. You use this information to debug design issues, especially for designs with a large number of macros. To query the preprocessing information about conditional compilation compiler directives such as `ifdef, set this variable to 1. To query about these and additionally macro expan- sions, set the variable to 2. The default is 0 i.e it will not log any information for either the preprocssing or macro expansion.

The informational messages can be found in the log by looking for the message code (FMR_VLOG-007). For example,

Information: Macro | append(...) | expanded to | x_master |. (File: example/example.sv Line: 8) (FMR_VLOG-007) Information: Rejecting MYDEF1 (File: example/example.sv Line: 3) (FMR_VLOG-007)

SEE ALSO

FMR_VLOG-007

hdlin_auto_netlist

Enables automatic use of the Verilog netlist reader by the read_verilog command.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to enable Formality to automatically try the Verilog netlist reader when no other reader is specified in the **read_verilog** command. If the Verilog netlist reader is not successful then the default reader is used.

To change the value of this variable, enter **set hdlin_auto_netlist** "*value*", where *value* is "true" or "false".

SEE ALSO

read_verilog(2)

hdlin_auto_top

Enables automatic detection and linking of the top-level design by read commands.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

Use this variable to enable automatic detection and linking of the top-level design by the read_verilog, read_vhdl, read_edif and read_db commands. When not enabled, you must determine which design to link using the existing set_top command.

If there are multiple top design candidates, then the read command finishes reading the designs but does not select a top design. Instead, an error message is issued. You can continue by issuing an explicit set_top command to specify the top design.

If there are multiple VHDL architectures/configurations for the top-level design, then the read command finishes reading the designs and links using the default architecture. A warning is issued that the default architecture/configuration is being used.

Note: If you do not want Formality to use the default architecture you must either include a configuration file or disabled auto-top mode and use an explicit set_top command that specifies the architecture.

To change the value of this variable, enter **set hdlin_auto_top** "*value*", where *value* is "true" or "false".

SEE ALSO

set_top(2)

hdlin_default_bbox_parameter_name

Controls the naming of parameters of black box for positional overrides.

TYPE

string

DEFAULT

P%d

DESCRIPTION

Formality issues an error if it cannot find a design to link with a specific cell. If User sets the variable "hdlin_unresolved_modules" to "black_box" then it generates black box designs in a library called "FM_BBOX" and then links the design to the cell.

By default, Formality generates blackbox design names using the reference design base-name. If variable "hdlin_unique_bbox_names" is set to true, Formality generates blackbox design names with the cell parameter information.

BBOX Parameter names for positional overrides will be generated based on the value set on the the variable "hdlin_default_bbox_parameter_name"

Default value of this variable is "P%d" %d in the value indicates position number starting from "1".

Hence by default the parameters will be named as P1, P2, P3...

Replace "P" in "P%d" with a valid identifier to change the parameter names.

EXAMPLE

Variables set:

set hdlin_unresolved_modules "black_box" set hdlin_default_bbox_parameter_name "PRM%d"

```
module top (output logic [1:0] out, input logic [1:0] in);
   mid #(1, 0) m1(out[1], in[1]);
endmodule
```

The blackbox design name generated for the cell m1 is : mid_PRM11_PRM20

SEE ALSO

hdlin_unresolved_modules(3)
hdlin_unique_bbox_names(3)

hdlin_define_synthesis_macro

This variable controls auto SYNTHESIS macro definition in verilog files

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

When *hdlin_define_synthesis_macro* is true, subsequent verilog code will be read as if the verilog statement *`define SYNTHESIS* exists before the code is read.

When the variable is false, the macro will not be automatically defined as above.

SEE ALSO

hdlin_ignore_translate(3)
hdlin_ignore_synthesis(3)

hdlin_do_inout_port_fixup

Specifies whether DDX-7 error messages will be generated at link time.

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to control whether DDX-7 error messages will be generated at link time.DDX-7 error messages occur when a net is driving an inout port that is also driving some other receiver. This variable exists to deal with cases where synthesis erroneously identifies DDX-7 error messages. It should always be set to its default "true" setting unless an erroneousDDX-7 error message has been found in a design. These error messages can be suppressed by setting this variable to "false".

Note: If you change the value of this variable, only subsequent link operations will be affected by the change.

To change the value of this variable, enter **set hdlin_do_inout_port_fixup** "*value*", where *value* may be "true" or "false".

hdlin_dw_sim_files_path

This variable specifies DesignWare simulation file path patterns.

TYPE

string

DEFAULT

"*/dw/sim_ver/vcs/*.v */dw/dw*/*/DW*_sim.vhd */dw/sim_ver/DW*.v */dw/dw0*/src_ver/*.v"

DESCRIPTION

Use this variable to specify additional DesignWare simulation file paths available outside "*hdlin_dwroot*". This variable allows you to set additional DesignWare simulation file paths apart from the existing default values. The Formality tool issues an error or warning message based on the variable "*hdlin_error_on_dw_sim_file_read*" if any DesignWare simulation files specified in **hdlin_dw_sim_files_path** are read.

To change the value of this variable, enter lappend **hdlin_dw_sim_files_path** "*value*". *value* is the new DesignWare simulation file path pattern.

SEE ALSO

hdlin_dwroot(3)
hdlin_error_on_dw_sim_file_read(3)
read_verilog(2)
read_sverilog(2)
read_vhdl(2)

hdlin_dwhomeip

Specifies one or more DesignWare Home IP Library install path.

TYPE

String

DEFAULT

""

DESCRIPTION

Use this variable to let Formality know the location of one or more DesignWare Home IP Library install path that contains the DesignWare components instantiated in any of the hierarchies read into the tool.

The **hdlin_dwhomeip** variable is a global variable set to the empty string "" by default, meaning that DesignWare instances will be left elaborated as black boxes. To enable resolving a DesignWare instance, set this variable to the one or more DesignWare Home IP Library install path containing the desired DesignWare components.

To change the value of this variable, enter **set hdlin_dwhomeip** "*value*", where *value* is something like "/u/designware-home/dwfc"

As DesignWare Home IP Library products have depedency over DesignWare Foundation library, **hdlin_dwroot** must be set to valid Synopsys Install root to update this TCL variable

SEE ALSO

hdlin_dwroot(3)

hdlin_dwroot

Specifies the DesignWare root.

TYPE

String

DEFAULT

""

DESCRIPTION

Use this variable to let Formality know the location of the Synopsys tree that contains the DesignWare components instantiated in any of the hierarchies read into the tool.

The **hdlin_dwroot** variable is a global variable set to the empty string "" by default, meaning that DesignWare instances will be left elaborated as black boxes. To enable resolving a DesignWare instance, set this variable to the top of the Synopsys tree containing the desired DesignWare components. The top of the Synopsys tree is the full path to the directory containing the DesignWare tree, as well as the packages and auxx directories.

To change the value of this variable, enter **set hdlin_dwroot** "*value*", where *value* is something like "/d/snps-1999.10"

hdlin_dyn_array_bnd_check

Controls whether logic is added to check the validity of array indices.

TYPE

String

DEFAULT

"Verilog"

DESCRIPTION

Controls whether logic is added to check the validity of array indices. The default is "Verilog". Enabling this variable for a language causes logic to be introduced to check the validity of array indices. This logic is designed to behave consistently with simulators in the case an array index is outside the array's bounds during an array write.

To change the value of this variable, enter set hdlin_dyn_array_bnd_check "value" where value is one of "Verilog", "VHDL", "None", or "Both"

```
"Verilog"  only Verilog files will have logic
           for out of range indices
"VHDL"     only VHDL files will have logic for
           out of range indices
"None"     Neither language will check for out of
           range indices
"Both"     Both Verilog and VHDL will have logic
           for out of range indices
```

Note: If you change the value of this variable, only designs read thereafter are affected by the change.

If set to "None" or the wrong language, the logic generated by Formality may not match simulation behavior in the out of range index cases.

For a list of **hdl** variables and their current values, type **printvar hdlin***.

hdlin_enable_assertions

Controls the interpretation of SystemVerilog assertions.

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

When the **hdlin_enable_assertions** variable is set to "false", Formality ignores SystemVerilog assertions. When this variable is set to "true", Formality converts certain assertions into design constraints during the **set_top** command.

To be converted to a constraint the assertion must belong to the supported subset for the current release. See the Design Compiler SystemVerilog User Guide for specifics about supported assertions. The assertion must also have a statement label and must be confirmed using the TCL **confirmed_SVA()** array variable:

```
set hdlin_enable_assertions true
read_sverilog -r my_file.sv
set confirmed_SVA(module1.scope1.assertion_label1) true
set confirmed_SVA(module1.global_assertion1) true
set_top module1
```

Note that assertion based constraints are only applied to reference designs. Any assertions in implementation designs will be ignored during verification.

To change the value of this variable, enter **set hdlin_enable_assertions** "*value*", where *value* is "true" or "false".

SEE ALSO

read_sverilog(2)

hdlin_enable_hier_naming

Generates hierachical instances and hierachical registers names for specific VHDL constructs.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable generates hierachical instances and hierachical registers names for the instances and registers under the following VHDL statements:

- VHDL GENERATE statement
- BLOCK statement
- PROCESS statement
- IF GENERATE statement

EXAMPLES

For the following VHDL generate block:

```
gen: for i in 0 to 1 generate

  proc1: process(clk)
    variable var : std_logic;
  begin
    if (clk'event and clk='1') then
      var := data(i);
 end if;
  end process;

  inst :  subblock port map(CLK => clk, ENABLE=> en, DATA => data(i));

end generate;
```

The generated register names are:

```
gen(0)/proc1/var_reg
gen(1)/proc1/var_reg
```

The generated instance names are:

```
gen(0)/inst
gen(1)/inst
```

Setting the **hdlin_enable_hier_naming** variable to **true** has the same effect as the following variables:

```
fm_shell > set hdlin_use_hierarchical_register_names true
fm_shell > set hdlin_use_vhdl_gen_hierarchy_for_naming true
```

SEE ALSO

hdlin_use_hierarchical_register_names(3)
hdlin_use_vhdl_gen_hierarchy_for_naming(3)

hdlin_enable_ieee_1735_support

TYPE

boolean

DEFAULT

true

DESCRIPTION

This variable enables some limited support in the read_(s)verilog command for the decryption of Verilog and SystemVerilog encrypted envelopes in the style of IEEE Std 1735-2014.

hdlin_enable_link_error_verbose

Enables verbose messages when link fails.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **hdlin_enable_link_error_verbose** variable provides additional information onFE-LINK-2 error. When cell cannot be linked to its reference design, this variable gives information on matching instance parameters with available set of designs in memory to facilitate the user.

To change the value of this variable, enter **set hdlin_enable_link_error_verbose** "*value*", where *value* is "true" or "false".

EXAMPLE

The following example shows how to use the **hdlin_enable_link_error_verbose** variable to enhance FE-LINK-2 error.

```
fm_shell (setup)> set hdlin_enable_link_error_verbose true
fm_shell (setup)> read_verilog -r -libname WORK test.v
Loading verilog file sub.v
Current container set to 'r'
1
fm_shell (setup)> read_power_model -r -libname WORK ref_model.fpm
Info: Loaded power model for design 'sub'.
1
fm_shell (setup)> set_top top
Setting top design to 'r:/WORK/top'
Status: Elaborating design top ...
Warning: Cannot link cell '/WORK/top/sub_design' to its reference design 'sub'. (FE-LINK-2)
Reason : Unable to match Parameters of instance with available set of designs in memory
Instance Parameters : p1=0
File : test.v (Line : 20)

Designs available in workspace:
---------------------------------
r:/WORK/sub
Design Parameters : p1=1
```

SEE ALSO

hdlin_unresolved_modules(3)

hdlin_enable_persistent_verilog_defines

Controls whether a macro definition from read command (read_verilog/read_sverilog) is to be preserved in-memory for successive read commands.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

Formality by default does not preserve macro-definitions over successive read commands. When hdlin_enable_persistent_verilog_defines app_option is enabled, macro-definitions from a read command (read_verilog/read_sverilog) is preserved in-memory for successive read commands. Only the macros defined in RTL will be preserved/persistent. Command line macro definitions (-define and -vcs +define+) will be removed from memory after executing one read command.

#test.v `define MAX 3

#test1.v module test(); logic [`MAX:0] x; endmodule

#fm.tcl read_sverilog -r test.v read_sverilog -r test1.v

Default Behavior (false):

- FM will throw read error since definition of macro MAX is not available in test1.v

Switch Enabled (true):

- FM will not throw read error since definition of macro MAX is preserved over multiple read commands.

DC/FC will provide the app_option in SVF through guide_environment. Formality will process this guidance and enable/disable the flow in SVF auto-setup flow with a warning if the guidance changes default/user setting of this app_option.

SEE ALSO

hdlin_enable_time_decl

It enables the support for time declaration.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable is used to enable the support for time declaration in verilog. However this the time units are not supported(THey are parsed and ignored).

EXAMPLE

This is an example test case, which is supported under the switch.

module top(); time netArray[3:0][4:0][5:0]; time data; always data=netArray[2][3]; endmodule

module top(); time data = 10ns; // here the unit ns is Ignored. endmodule

hdlin_enable_upf_compatible_naming

Controls the RTL object naming style settings to make it easier to apply the same UPF file across multiple tools at the RTL level.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls the RTL object naming style settings to make it easier to apply the same UPF file across multiple tools at the RTL level. This makes it conform more closely to language standard naming conventions for RTL generate statements, VHDL records and System Verilog structures.

Set the **hdlin_enable_upf_compatible_naming** to **true** to ignore the current values of the following variables and has effects that match the following variable settings:

```
fm_shell > set hdlin_enable_hier_naming true
fm_shell > set hdlin_field_naming_style "%s.%s"
```

Set the **hdlin_enable_upf_compatible_naming** is set to the default **false** to use the current settings of these variables.

SEE ALSO

hdlin_enable_hier_naming(3)
hdlin_field_naming_style(3)

hdlin_enable_verilog_assert

Controls whether SystemVerilog ASSERT (and related) statements in Verilog 95 and 2001 source files are treated as errors (default) or ignored.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

Formality's Verilog RTL synthesizer can be configured to accept SystemVerilog assert and assert-related statements, ignoring them without generating syntax errors. When **hdlin_enable_verilog_assert** is set "false" (default) Formality considers SystemVerilog "assert" statements in Verilog 95 and 2001 source files as errors. The assert-related keywords (e.g. assert, sequence, property, ...) may appear in 95 and 2001 source files as user variables and function names. When set to "true" Formality will parse assert related statements according to the SystemVerilog standard. These statements must be syntactically correct and are simply ignored.

Note: If you change the value of this variable, only future read operations will be affected by the change.

To change the value of this variable, enter **set hdlin_enable_verilog_assert** "*value*", where *value* is "true" or "false".

hdlin_enable_verilog_configurations

Enables verilog configuration support.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables elaborating a design based on verilog configuration rules. When true, auto-netlist flow gets disabled. Users would then need to explicitly use "read_verilog -netlist" for reading implementation netlist.

SEE ALSO

hdlin_enable_verilog_configurations_array_n_block(2)

hdlin_enable_verilog_configurations_array_n_block

This variable enables configuration support for "Array of instance" and non-LRM configuration syntax.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable enables below two features for verilog configurations.

1) Allows to use brackets "[ ]" without any preceding escape id for an instance clause(This is non-LRM syntax). Example : instance top.genblk[0].a liblist aLib

2) Supports instance clause used on individual bits of an array of instance in configuration. Example : instance top.inst_arr[1] liblist aLib instance top.inst_arr[0] liblist bLib instance top.inst_arr[2] liblist cLib

where "inst_arr" is an instance array. Here all element(s) of an instance array will be configured based on first instance clause "instance top.inst_arr[1] liblist aLib" and remaining instance clauses will be ignored for that perticular array instance.

SEE ALSO

hdlin_enable_verilog_configurations(2)

hdlin_error_on_duplicate_design

Controls the severity of messages that alert the user to possible overwrite of a design when it analyzes a duplicate design

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to control the severity of messages that alert the user to possible overwrite of a design when it analyzes a duplicate design. To treat these messages as errors, the value of this variable should be "true". To treat these messages as warnings the value of this variable should be "false".

This variable is a global variable set to "false" by default, meaning that whenever user reads in a duplicate design, Formality will replace existing design with latest version of the design by flagging a message.

Note: If you change the value of this variable, only future read and link operations will be affected by the change.

To change the value of this variable, enter **set hdlin_error_on_duplicate_design**"*value*", where *value* is "true" or "false".

hdlin_error_on_dw_sim_file_read

Controls the severity of the message that alerts you when the Formality tool reads a DesignWare simulation file.

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

Use this variable to control the severity of the message that alerts you when the tool reads a DesignWare simulation file. By default, the Formality tool issues a warning when you read a DesignWare simulation file. By setting this variable to "true", the Formality issues the error "FMR_ELAB-536" at the beginning of set_top and exits

To change the value of this variable, enter **set hdlin_error_on_dw_sim_file_read** "*value*", where *value* is "true" or "false".

SEE ALSO

hdlin_dw_sim_files_path(3)
hdlin_dwroot(3)
read_verilog(2)
read_sverilog(2)
read_vhdl(2)

hdlin_error_on_supply_type_port

Enables a check that performed on each design to identify a supply0/supply1 type port.

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to control the check that made on each design to identify a supply0/supply1 type port. A Supply type port is not illegal in Verilog/SystemVerilog, but it results in a netlist that will not verify correctly becuase all of the ports cannot toggle as expected. It also makes the verification results confusing and difficult to debug after verification.

For example, below Netlist leads to an error in Formality.

module test (out, a,b); output out; input a, b; supply0 a; // ERROR: net_type for port ‘a’ is supply0 ... endmodule

This variable is a global variable set to "true" by default, meaning that Formality flashes an error message when a port declared as supply0/supply1. To reverse this effect, you can set this variable to "false" so that Formality skips this check. We recommend not altering this variable from its "true" default state for UPF designs. Verification results are very difficult to debug and performance may be significantly impacted if you disable this check.

EXAMPLE

The following example shows how to use the variable during set_top.

```
fm_shell (setup)> set hdlin_error_on_supply_type_port true
fm_shell (setup)> read_verilog -r test.v
No target library specified, default is WORK
Loading verilog file 'test.v'
Current container set to 'r'
1
fm_shell (setup)> set_top test
Setting top design to 'r:/WORK/test'
Status:  Implementing inferred operators...
Error: Invalid type ‘supply0’ specified for port 'a' in design 'test' (FM-563)
Error: Failed to set top design to 'r:/WORK/test' (FM-156)
0
fm_shell (setup)> set hdlin_error_on_supply_type_port false
false
fm_shell (setup)> set_top -a
Setting top design to 'r:/WORK/test'
```

```
Status:  Implementing inferred operators...
Top design successfully set to 'r:/WORK/test'
Implementation design set to 'r:/WORK/test'
```

hdlin_error_on_unresolved_svf_netlist_cell_reference

Enables early termination of svf processing when Formality detects unresolved cell references in SVF netlists.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

When enabled, svf processing will stop and return if Formality detects any unresolved cell references in SVF netlists. You can ignore the unresolved references by setting this var to false and continue processing guidances. However, it may lead to Hard Verification due to some SVF datapath rejections.

SEE ALSO

FM-595(n)

hdlin_exclude_from_techlib

Reads designs in a technology library in as standard (non-technology) designs.

TYPE

string

DEFAULT

" "

DESCRIPTION

This hidden variable is intended for Synplicity flow only.

When used with the **hdlin_library_directory** variable, designs in the specified directory are read in as technology designs. the **hdlin_exclude_from_techlib** variable interprets designs in the directory specified using the **hdlin_library_directory** variable as standard (non-technology) designs.

By default, design in the directory specified using the hdlin_library_directory variable are considered technology cells.

To change the value of this variable, enter **set hdlin_exclude_from_techlib** "*value*", where *value* is a regular expression (regexp).

EXAMPLE

In the following example, the Formality reader interprets designs in the $XILINX/verilog/xeclib/unisims directory that start with either "mydzn" or "DSP48" as standard (non-technology) designs. Note that {non-}technology interpretation in Formality happens at a file level. So, if designs that start with "mydzn" in the bar.v file are interpreted as standard designs.

```
set hdlin_library_directory $XILINX/verilog/xeclib/unisims
set hdlin_exclude_from_techlib "Foo* DSP48*"
```

SEE ALSO

hdlin_library_directory(3)

hdlin_field_naming_style

Defines the parts of the net names that the tool generates corresponding to the fields in VHDL records or in the SystemVerilog structs.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable defines the parts of the net names that the tool generates corresponding to the fields in VHDL records and SystemVerilog structs.

By default the **hdlin_field_naming_style** variable is not set and it is derived from the **bus_naming_style** and **bus_dimension_separator_style** variables. If the **bus_naming_style** and **bus_dimension_separator_style** variables are set to **%s[%d]** and **][**, or **%s<%d>** and **><**, it is possible to specify the **hdlin_field_naming_style** variable as "%s<%s>", "%s[%s]", or "%s.%s",

in which the first %s stands for the name up to the field and the second %s stands for the field.

The **hdlin_field_naming_style** variable must be of the form "%sX%s" or "%sX%sY", where X and Y are (possibly identical) non-whitespace characters. Only the record fields are named using the **hdlin_field_naming_style** variable. The array elements are named using the **bus_naming_style** variable.

If the **bus_naming_style** variable is set to "%s<%d>" and the **bus_dimension_separator_style** variable is set to anything other than "><", the **hdlin_field_naming_style** variable is ignored and a warning message is displayed.

EXAMPLES

Formality variables:

```
set bus_naming_style "%s<%d>"
set bus_dimension_separator_style "><"
set hdlin_field_naming_style "%s.%s"
```

RTL:

```
package packg is
   type type1 is record
      FIELD : integer range 0 to 3;
   end record;
end packg;
```

```
use WORK.packg.all;
entity test is
  port (clk   : in bit;
        out1  : buffer type1);
end;

architecture test of test is
begin
process (clk)
begin
    if (clk'event and (clk = '1')) then
          out1.FIELD <= 3;
    end if;
end process;

end test;
```

The registers for the VHDL record element FIELD will be named as out1_reg.FIELD<0> and out1_reg.FIELD<1>.

SEE ALSO

bus_naming_style(3)
bus_dimension_separator_style(3)
hdlin_enable_upf_compatible_naming(3)

hdlin_filter_physical_only_cells

Enables removal of physical_only cells during set_top.

TYPE

string

DEFAULT

"false"

DESCRIPTION

hdlin_filter_physical_only_cells allows users to filter (remove) physical_only cells from the design during set_top.

The tool identifies physical_only technology library cells in two ways :

- DB(.db) cells which has any of the following attributes: *is_decap_cell, is_filler_cell, is_tap_cell, antenna_diode_type, bump_cell*
- Any technology library cell with a name matching a value specified in the variable **hdlin_physical_only_cells**, and that cell does not contain any logic signal pins/ports

Cells identified as physical_only cells will be removed from the design during design set_top. Info messages (in formality.log) contain details of which phyical_only cells were filtered (removed).

SEE ALSO

hdlin_physical_only_cells(2)

hdlin_hierarchy_separator_style

Specifies the separator character used in the path names of hierarchical instance and register names.

TYPE

string

DEFAULT

"/"

DESCRIPTION

This variable specifies the separator character used in path names when instance and register names are generated using one of the following variables,

- **hdlin_use_vhdl_genl_hierarchy_for_naming**
- **hdlin_use_hierarchical_register_names**

EXAMPLE

```
set hdlin_hierarchy_separator_style "."

gl1: for i in 0 to 1 generate
  u1 : etff port map(CLK => clk, ENABLE=> en, DATA => data(j*2+i), MUXDATA => q(j*2+i));
end generate;

Names of the instances generated would be
        gl1(0).u1
        gl1(1).u1
```

SEE ALSO

hdlin_use_vhdl_gen_hierarchy_for_naming(3)
hdlin_use_hierarchical_register_names(3)

hdlin_ignore_builtin

Sets the option to ignore or not ignore the built-in directive in VHDL files.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to ignore or not ignore the built-in directive in VHDL files. If you want to ignore thebuilt-in directive, set the value to "true".

To change the value of this variable, enter **set hdlin_ignore_builtin** "*value*", where *value* is "true" or "false".

If you change the value of this variable, only VHDL files you read in thereafter are affected by the change.

hdlin_ignore_dc_script

Specifies to ignore or not ignore compiler directives **pragma dc_script_begin** and **pragma dc_script_end** directives in Verilog or VHDL files.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to ignore or not ignore the **pragma dc_script_begin** and **pragma dc_script_end** directives in Verilog or VHDL files. If you want to ignore these directives, set the value to "true".

To change the value of this variable, enter **set hdlin_ignore_dc_script** "*value*", where *value* is "true" or "false".

If you change the value of this variable, only Verilog or VHDL files you read in thereafter are affected by the change.

hdlin_ignore_embedded_configuration

Specifies to ignore or not ignore the **embedded configurations** in VHDL files.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

Use this variable to ignore or not ignore the **embedded configurations** in VHDL files.

To change the value of this variable, enter **set hdlin_ignore_embedded_configuration** "*value*", where *value* is "true" or "false".

hdlin_ignore_full_case

Specifies to ignore or not ignore the **full_case** directive in Verilog files.

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to ignore or not ignore the **full_case** directive in Verilog files.

To change the value of this variable, enter **set hdlin_ignore_full_case** "*value*", where *value* is "true" or "false".

Note: If you change the value of this variable, only Verilog files you read in thereafter are affected by the change.

hdlin_ignore_label

Specifies to ignore the **label** directive in VHDL files.

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to ignore the **label** directive in VHDL files. If you want to ignore the directive, set the value to "true".

To change the value of this variable, enter **set hdlin_ignore_label** "*value*", where *value* is "true" or "false".

Note: If you change the value of this variable, only VHDL files you read in thereafter are affected by the change.

hdlin_ignore_label_applies_to

Specifies to ignore or not ignore the **label_applies_to** directive in Verilog or VHDL files.

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to ignore or not ignore the **label_applies_to** directive in Verilog or VHDL files. The default is set to "true" so the directive is ignored. To reverse this effect, you can set this variable to "false".

To change the value of this variable, enter **set hdlin_ignore_label_applies_to** "*value*", where *value* is "true" or "false".

Note: If you change the value of this variable, only Verilog or VHDL files you read in thereafter are affected by the change.

hdlin_ignore_map_to_module

Specifies to ignore or not ignore the **map_to_module** attribute in Verilog or VHDL files.

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to ignore or not ignore the **map_to_module** attribute in Verilog or VHDL files. The default is set to "true" so the attribute is ignored. To reverse this effect, you can set this variable to "false".

To change the value of this variable, enter **set hdlin_ignore_map_to_module** "*value*", where *value* is "true" or "false".

Note: If you change the value of this variable, only Verilog or VHDL files you read in thereafter are affected by the change.

hdlin_ignore_map_to_operator

Ignores the **map_to_operator** attribute in Verilog or VHDL files.

TYPE

string

DEFAULT

"false"

DESCRIPTION

The variable ignores the **map_to_operator** attribute in Verilog or VHDL files when you set the variable to *true*. By default, the variable is set to *false*.

To change the value of the variable, use the **set hdlin_ignore_map_to_operator** *value* variable. Where *value* is *true* or *false*.

Note: If you change the value of the variable, it affects only those Verilog or VHDL files that are read after changing the value of the variable.

hdlin_ignore_parallel_case

Specifies to ignore or not ignore the **parallel_case** directive in Verilog files.

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to ignore or not ignore the **parallel_case** directive in Verilog files. The default is set to "true" so the directive is ignored. To reverse this effect, you can set this variable to "false".

To change the value of this variable, enter **set hdlin_ignore_parallel_case** "*value*", where *value* is "true" or "false".

Note: If you change the value of this variable, only Verilog files you read in thereafter are affected by the change.

hdlin_ignore_resolution_method

Specifies to ignore or not ignore the **resolution_method** directive in VHDL files.

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to ignore or not ignore the **resolution_method** directive in VHDL files. The default is set to "true" so the directive is ignored. To reverse this effect, you can set this variable to "false".

To change the value of this variable, enter **set hdlin_ignore_resolution_method** "*value*", where *value* is "true" or "false".

Note: If you change the value of this variable, only VHDL files you read in thereafter are affected by the change.

hdlin_ignore_synthesis

Specifies to ignore or not ignore the **synthesis_off** and **synthesis_on** directives in Verilog files.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to ignore or not ignore the **synthesis_off** and **synthesis_on** directives in VHDL or Verilog files. The default is set to "false" meaning that synthesis off and synthesis on directives are not ignored. To reverse this effect, you can set this variable to "true".

To change the value of this variable, enter **set hdlin_ignore_synthesis** "*value*", where *value* is "true" or "false".

Note: If you change the value of this variable, only Verilog or VHDL files you read in thereafter are affected by the change.

hdlin_ignore_translate

Specifies to ignore or not ignore the **translate_off** and **translate_on** directives in Verilog or VHDL files.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to ignore or not ignore the **translate_off** and **translate_on** directives in Verilog or VHDL files. The default is set to "false" meaning that these directives are not ignored. To reverse this effect, you can set this variable to "true".

To change the value of this variable, enter **set hdlin_ignore_translate** "*value*", where *value* is "true" or "false".

Note: If you change the value of this variable, only Verilog or VHDL files you read in thereafter are affected by the change.

hdlin_infer_function_local_latches

Controls whether to infer latches inside functions

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

Use this control variable to infer the latches inside functions and tasks.

When the value of this variable is true, latches inside function and task bodies can be inferred, according to the same rules used in always blocks.

When the value is false (the default), no latches are inferred in func- tions or tasks.

hdlin_interface_only

Enables creation of a design(s) consisting only of an interface.

TYPE

string

DEFAULT

""

DESCRIPTION

Use this variable to direct Formality's Verilog, VHDL, DB, and EDIF readers to produce interface-only designs. Resulting designs consist only of the source design's specified port and parameter declarations. The remaining design internals are either not synthesized (in the case of Verilog or VHDL behavioral RTL) or discarded (for Verilog netlist, DB, and EDIF based designs).

Formality design readers compare the name of the design it is currently processing to the value of this variable. If a match is found, the design is converted to an interface-only design. Compare rules are based on TCL's glob-style pattern matching syntax.

In case of VHDL if an architecture is specified as an interface only using this variable then you need to end the architecture in one of the following ways:

end architecture;

OR

end architecture <architecture_name>;

OR

end <architecture_name>;

To change the value of this variable, enter **set hdlin_interface_only** "*value*", where *value* is in TCL glob-style pattern matching form.

hdlin_keep_feedback

Controls whether or not the combinational feedback loops are retained or removed by Formality.

TYPE

Boolean

DEFAULT

false

ENABLED SHELL MODES

all

DESCRIPTION

This variable controls whether statements like SigA = SigA generate a combinational feedback loop or generate an asynchronous load latch without state feedback. If it is required to retain the feedback loops, the **value** of variable should be set to 'true'.

Example:

entity mytest is port(Inp: in std_logic; OutP: out std_logic); end mytest;

architecture top of mytest is signal SigA: std_logic; begin test: process (Inp, SigA) begin if Inp='0' then SigA <='1'; else SigA <=SigA; end if; end process; OutP <= SigA; end top;

This variable causes Formality not to create feedback loops by default and match Design Compilers default behavior, hence this variable should only be set to true if the same variable in Design Compiler is also set to true.

hdlin_library_assume_switch_pdf

Specifies whether Formality adds an assumed power down function to power switch outputs

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

There are versions of Library Compiler that do not allow power down functions to be specified for switched outputs of type *internal_power* or *internal_ground*. This means that there is no way to specify the behavior of a switch output when a non-switched supply input changes state.

When this variable is set to *true* Formality will add a power down function such that when any non-switched supply of type *primary_power* or *primary_ground* is off the switch output will be in an "unknown" state.

hdlin_library_attribute_interface_only

Controls whether library cells that have black-box attributes are created as 'interface_only'.

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

Use this variable to control whether Formality creates 'interface_only' designs for library cells that contain black-box attributes as defined by the **hdlin_library_preserve_bbox_attributes** variable.

Note: If you change the value of this variable, only libraries read thereafter are affected by the change.

To change the value of this variable, enter **hdlin_library_attribute_interface_only** "*value*", where *value* may be "true" or "false".

SEE ALSO

hdlin_library_preserve_bbox_attributes(3)
hdlin_interface_only(3)

hdlin_library_auto_correct

Specifies whether simple errors in library power behavior are automatically corrected.

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

When this variable is set to *true*, the tool attempts to correct simple errors in technology library cell power models. Example: It may add power-down functions to output pins that are not constrained.

SEE ALSO

report_libraries(2)
hdlin_library_ignore_errors(3)

hdlin_library_auto_correct_pg_pin_direction

Auto corrects direction of **inout** pg_pins based on the usage inside the technology library cell.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

When this variable is set to *true*, the tool attempts to correct the direction of**inout** pg_pin based on the usage inside technology library cell. If the net connected to pg_pin has only readers, then changes the direction to **in** and if the net connected to pg_pin has only drivers, then changes the direction to **out**.

SEE ALSO

report_libraries(2)
hdlin_library_auto_correct(3)

hdlin_library_cell_merge_port

Specifies a list of library cell port names that should be considered as the 'Q' output of the merged cell during the merging of tech cell registers instantiated in the Verilog Model.

TYPE

String

DEFAULT

""

DESCRIPTION

This variable should be set during setup before reading libraries or designs. The syntax to use the variable is

set hdlin_library_cell_merge_port {cell_ports}

where cell_ports are of the form "cell_name/port_name". The default value of the variable is empty string "". By default, the technology cell registers are merged based on the order, the registers are instantiated in the Verilog model.

Example 1: Assume the output of the library cell named "MYREG" has a port named "MYQ" that should be the 'Q' output of the merged cell.

set hdlin_library_cell_merge_port {MYREG/MYQ}

Example 2: Assume the output of all library cells that start with the name "REG" have a port named "MYQ" that should be the output of the merged cell.

set hdlin_library_cell_merge_port {REG*/MYQ}

Example 3: Assume the output of the library cell named "MYREG" has a port named "MYQ" that should be the 'Q' output of the merged cell and another library cell named "OTHERCELL" has a port named "OQ" that should be the 'Q' output of the merged cell.

set hdlin_library_cell_merge_port {MYREG/MYQ OTHERCELL/OQ}

hdlin_library_change_analog_inout_to_input

Specifies whether Formality changes analog library cell pins of direction *inout* to *input*

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

When this variable is set to *true*, library cell pins of direction *inout* that are marked with the attribute "is_analog" will be changed to direction *input*.

hdlin_library_directory

Designates all designs contained within the directory(ies) as TECHNOLOGY designs.

TYPE

string

DEFAULT

DESCRIPTION

Use this variable to declare a set of directories as containing only TECHNOLOGY designs. This variable provides the same functionality as the read_verilog command's -technology_library switch only on a directory basis. All designs within a directory specified by this variable will be loaded into a technology library.

The -libname switch takes precedence over the hdlin_library_directory variable. This implies that when the read_verilog command contains the -libname switch, all designs will reside in the specified library, regardless of whether it is a technology library or user design library.

To change the value of this variable, enter **set hdlin_library_directory** "*value*", where *value* is a space-delimited list of directories. These directories may either be full pathnames or relative to the specified search_path.

SEE ALSO

hdlin_library_file(3)

hdlin_library_file

Designates all designs contained within the file(s) as TECHNOLOGY designs.

TYPE

string

DEFAULT

DESCRIPTION

Use this variable to declare a set of files as containing only TECHNOLOGY designs. This variable provides the same functionality as the read_verilog command's -technology_library switch only on a file by file basis. All designs within a file specified by this variable will be loaded into a technology library.

The -libname switch takes precedence over the hdlin_library_file variable. This implies that when the read_verilog command contains the -libname switch, all designs will reside in the specified library, regardless of whether it is a technology library or user design library.

To change the value of this variable, enter **set hdlin_library_file** "*value*", where *value* is a space-delimited list of files. These files may either be full pathnames or relative to the specified search_path.

SEE ALSO

hdlin_library_directory(3)

hdlin_library_ignore_errors

Specifies whether errors in linked library cells cause commands to fail.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

When this variable is set to *false*, the **set_top** and **load_upf** commands fail if any of the technology library cells that are linked to the design have incorrect or incomplete power models. When the variable is set to *true* the errors are ignored.

SEE ALSO

report_libraries(2)
hdlin_library_auto_correct(3)

hdlin_library_interface_only_corruption

Preserve power corruptions in interface only library cells read using **read_db**.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

By default, Formality preserves power corruption in interface only DB cells read using **read_db**.
When this variable is *false*, during **read_db** Formality creates an interface only blackbox and does not preserve power corruptions for the pg (power aware) version (with suffix #PWR and #UPF) of the library cells. This variable has no effect on non-pg (non power aware) version of the library cell.

SEE ALSO

read_db(2)
hdlin_interface_only(3)

hdlin_library_preserve_bbox_attributes

Preserves the db attributes provided in the list which are set on the db libcells.

TYPE

string

DEFAULT

"pad_cell is_macro_cell"

DESCRIPTION

Use this variable to specify list of attributes that needs to be preserved in netlist during read_db operation.

hdlin_library_report_summary

This variable controls whether a warning summary is produced for Verilog library files.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to enable Formality to report a summary of the warnings produced when reading or analyzing Verilog library files. The individual warning messages sent to the "formality.log" file are not controlled by this variable.

To change the value of this variable, enter **set hdlin_library_report_summary** "*value*", where *value* is "true" or "false".

hdlin_library_warn_on_missing_power_down_attributes

Specify one or more library cells whose modelling errors will be downgraded and reported as warnings.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable is used to change the tools library checking Errors to Warnings.

The variable value is a string of library cell names and can contain TCL glob-style patterns to match multiple cells.

During library checking, if any matching library cell name is found, the library model Errors are downgraded and reported as Warnings as part of **Library Checking Summary** and **report_libraries -defects**. No messages are generated for specified names that do not match valid library cell names. This variable provides the user with granular control for waiving errors on specified cells and is independent of the tool variable **hdlin_library_ignore_errors**.

EXAMPLE

prompt> set hdlin_library_warn_on_missing_power_down_attributes "DIODE*X* VCHECK1* PSNIFF13AX"

SEE ALSO

hdlin_library_ignore_errors(3)
report_libraries(2)

hdlin_link_ignore_dw_sim_models

Controls linking DesignWare root library designs with identically-named user designs in user libraries.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

You can set the **hdlin_dwroot** variable to a valid synthesis install path. This allows the Formality tool to automatically resolve any instances of DesignWare components found in RTL. If you reads in DesignWare files directly, the tool picks the user-read component ahead of the **hdlin_dwroot** version. You can read simulation or different synthesis version of DesignWare leading to verification challenges. To avoid this issue, utilize this option to notify the Formality tool to link to the **hdlin_dwroot** version ahead of user library versions.

To change the value of this variable, enter **hdlin_link_ignore_dw_sim_models** "*value*", where *value* can be "true" or "false".

SEE ALSO

hdlin_dw_sim_files_path(3)
hdlin_dwroot(3)
read_verilog(2)
read_sverilog(2)
read_vhdl(2)

hdlin_link_portname_allow_period_to_match_underscore

Enables the linker to allow a period (.) as an alternative to an underscore (_) when doing a SystemVerilog interface type port name matching.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

During linking, if named port mapping is used in the cell instance statement, the linker resolves port connections based on the port names. If the linker is unable to find a matching port name, and the hdlin_link_portname_allow_period_to_match_underscore variable is set to true, the linker replaces the "." characters in the port name from the cell instance statement with the "_" character to see if there is a match. This variable has no effect when positional port mapping is used instead.

This variable has no effect on non-interface type port connections.

hdlin_link_portname_allow_square_bracket_to_match_underscore

Enables the linker to allow a square bracket([]) as an alternative to an underscore (_) when doing port name matching.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

During linking, the linker resolves port connections based on port names when named port mapping is used in cell instantiation. In the default mode, the linker looks for an exact match of the port names. Setting the hdlin_link_portname_allow_square_bracket_to_match_underscore variable to true, allows the linker to also match the "[]" characters with the "_" character. This variable has no effect when positional port mapping is used. This variable should be used only for matching modport arrays. Formality will assume that the array of modport in the Subblock is in canonical form ([0:N]).

SEE ALSO

hdlin_link_portname_allow_period_to_match_underscore(3)

hdlin_merge_parallel_switches

Specifies whether functionally equivalent switch cells are automatically merged.

TYPE

Boolean

DEFAULT

"true"

ENABLED SHELL MODES

Setup

DESCRIPTION

As part of power network synthesis, implementation tools might expand a single UPF power switch into thousands of coarse grain switch cells in a variety of functionally equivalent configurations. Having large numbers of switches driving the same supply nets causes performance problems during verification and makes debugging difficult.

When the **hdlin_merge_parallel_switches** variable is set to *true*, the tool looks for nets driven by multiple coarse grain switch cells during the *set_top* command and merges switches that it can prove are equivalent.

After equivalent switches are merged, the tool adds messages to the log file indicating which supply nets are affected and how many of the driving switches are eliminated.

To preserve the netlist in its unmerged form, set this variable to *false* before setting the top design.

SEE ALSO

set_top(2)

hdlin_multiplier_architecture

Defines the architecture generated for multiplier or DW02_multp instances. This variable will only affect results if it is changed before a design file is read.

TYPE

string

DEFAULT

none

DESCRIPTION

Use this variable to help define the architecture Formality will generate for multiplier or DW02_multp instances encountered in the reference design when multiplier generation is enabled (see **enable_multiplier_generation**)

Before issuing a read command, change the value of this variable as follows:

set hdlin_multiplier_architecture *value*

where *value* is none, csa, nbw, wall or dw_foundation:

* none - No multiplier architecture is specified with **hdlin_multiplier_architecture**.

* csa - Generate a csa architecture for multiplier instances. Invalid value when used with DW02_multp.

* nbw - Generate an nbw architecture for multiplier or DW02_multp instances.

* wall - Generate a wall architecture for multiplier or DW02_multp instances.

* dw_foundation - Attempt to choose the same architecture Design Compiler was likely to have chosen for each multiplier instance. If the combined widths of the multiplier inputs is less than or equal to \fbdw_foundation_threshold, generate an nbw architecture. Otherwise, generate a wall architecture.

This variable will not have any affect unless **enable_multiplier_generation** is set to true.

SEE ALSO

report_architecture(2)
set_architecture(2)
enable_multiplier_generation(3)
architecture_selection_precedence(3)
dw_foundation_threshold(3)

hdlin_normalize_blackbox_busses

Controls how Formality names busses of black boxes during the link operation.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to control how Formality names busses of black boxes during the link operation. The default variable is set to "false", meaning that Formality should not change the names of black box busses during the link operation. If the bus signal names should be "normalized" to have indexes from WIDTH-1 down to 0, then set this variable to "true".

Note: If you change the value of this variable, only designs linked thereafter are affected by the change.

To change the value of this variable, enter **set hdlin_normalize_blackbox_busses** "*value*", where *value* may be "true" or "false".

hdlin_physical_only_cells

Enables **set_top** to successfully link a design, which contains some missing cell definitions.

TYPE

string

DEFAULT

""

DESCRIPTION

Use this variable to direct Formality's Verilog, VHDL, and DB readers to produce a successfully linked design when containing some missing cell (down design) definitions. The resulting design will consist of black boxes where a missing cell definition was encountered if the cell name has been specified in this variable.

Formality design readers compare the name of the design it is currently processing to the value of this variable. If a match is found, the design is converted to a black box design. Compare rules are based on TCL's glob-style pattern matching syntax.

To change the value of this variable, enter **set hdlin_physical_only_cells** "*value*", where *value* is in TCL glob-style pattern matching form.

SEE ALSO

hdlin_unresolved_modules(3)

hdlin_power_config_db_library

This variable can be used to specify a list of DB library files that should be used to map power and non-power(multibit) information to Verilog library cells.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable can be used to specify a list of DB library files that should be used to map power and non-power(multibit) information to Verilog library cells. The list can be pathnames to files. If they are not rooted pathnames, then the search_path will be used to find the files. Existing .db libraries read with read_db will not be used for mapping power attributes to the Verilog library cells.

EXAMPLE

```
fm_shell (setup)> set hdlin_power_config_db_library \
                  { db_file1.db db_file2.db /disk1/libs/db_file3.db }
```

hdlin_preserve_assignment_direction

Specifies how to control the assignment direction for your Verilog description

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

Formality does not preserve assignment direction in Verilog RTL and netlist designs. This variable helps in preserving those buffers used in the assignment.

To change the value of this variable, enter **set hdlin_preserve_assignment_direction** "*value*", where *value* is "true" or "false".

hdlin_preserve_empty_modules

This variable prevents black boxing of empty modules given with it

TYPE

string

DEFAULT

""

DESCRIPTION

Formality creates black box for empty modules.Use this variable to direct Formality's Verilog readers to prevent back boxing empty designs.Formality design readers compare the name of the empty design it is currently processing to the value of this variable. If a match is found, the design will not be converted to black box.Compare rules are based on TCL's glob-style pattern matching syntax.

To change the value of this variable, enter **set hdlin_preserve_empty_modules** "*value*", where *value* is in TCL glob-style pattern matching form.

hdlin_sv_auto_decl_inits

Controls whether variables are initialized, possibly with default values, according to the 1800 standard.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

This variable is used to control whether the Formality RTL SystemVerilog reader conforms to the 1800 standard with respect to automatic variable initializations.

To change the value of this variable, enter **set hdlin_sv_auto_decl_inits** "*value*", where *value* may be "true" or "false".

* "true" - Uses 1800 standard

* "false" - Does not use 1800 standard

SEE ALSO

read_sverilog(2)

hdlin_sv_blackbox_modules

Specify one or more SystemVerilog modules to be ignored during design read.

TYPE

string

DEFAULT

""

DESCRIPTION

The hdlin_sv_blackbox_modules allows the user to specify one or more SystemVerilog modules to be ignored during design read.

The modules that are to be ignored are specified by setting the variable hdlin_sv_blackbox_modules to a list of modules. For Example :

prompt> set hdlin_sv_blackbox_modules dff

Here dff is the name of module to be ignored, as coded in the below RTL.

```
module dff (input clk, input d, output q);
input clk, d;
output q;
reg q;
always @(posedge clk) q = d;
endmodule

module top;
reg data, clock;
wire q_out, net_1;
dff inst_1 (.d(data), .q(net_1), .clk(clock));
dff inst_2 (.clk(clock), .d(net_1), .q(q_out));
endmodule
```

To change the value of this variable, enter set hdlin_sv_blackbox_modules "value",where value is in TCL glob-style pattern matching form.

Formality Sverilog reader compares the name of the module it is currently reading to the value of this variable. If a match is found, the module is ignored. Compare rules are based on TCL's glob-style pattern matching syntax.

The warning message (FMR_VLOG-931) is generated during read_sverilog as shown below

Info: mod1.v:2: The declaration of module 'mod1' is being ignored,because the module name is in hdlin_sv_blackbox_modules. (FMR_VLOG-931)

If the tool tries to link the modules (during set_top command), linker error is generated.

No messages are generated for specified names that do not match valid module names.

SEE ALSO

read_sverilog(2)

hdlin_sv_packages

Specifies how System Verilog packages should be analyzed.

TYPE

string

DEFAULT

"dont_chain"

DESCRIPTION

Specifies which, if any, semantics the read_sverilog command should apply when a SystemVerilog source file declares a package. All settings accept "packages" (declarations, references and imports) as specified in section 19.2 of the IEEE-1800-2005 System Verilog standard. The allowed values for hdlin_sv_packages are dont_chain (the default) and chain.

The default behavior is as described in Clause 26 of IEEE Std 1800-2012, the standard for SystemVerilog. Overriding the default to chain changes how an import statement treats names imported into the topmost (global) scope of a package_declaration.

To change the value of this variable, enter **set hdlin_sv_packages**"*value*".

* "dont_chain" - Enforces the IEEE standard and prevents imported names from being re-exported to clients of the package being declared.

* "chain" - Always re-exports names that are imported into the global scope of a package. An imported name and its definition which are re-exported ("chained" in VCS parlance) will not collide or interfere with itself in those cases where several intermediate packages redistribute content they acquired from a common source package.

SEE ALSO

read_sverilog(2)

hdlin_sv_port_name_style

Controls the naming of ports that are of a complex data type (i.e. MDA, Structs, Unions) in a SystemVerilog design.

TYPE

string

DEFAULT

"complex"

DESCRIPTION

This variable is used to modify the style of complex data type port names created in the reference System Verilog design to match correctly with the port names of the implementation design. Depending on how the implementation design is generated, the complex port names can be a single packed vector or a name based on the multiple dimensions and/or field names. A failing verification can occur due to bad matching if this variable is not set properly to match the implementation design.

To change the value of this variable, enter **set hdlin_sv_port_name_style** "*value*", where *value* may be "vector" or "complex".

* "vector" - Generates the name for the complex data port as if it is a single packed vector with the bounds being the bitwidth of the complex type minus one downto zero.

* "complex" - Uses the internal net name that has all the multiple dimensions and/or field name information as the port name.

SEE ALSO

read_sverilog(2)

hdlin_sv_union_member_naming

Controls the naming styles for elements associated with the union data type in SystemVerilog.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

By default, the tool uses simple names for elements inferred from unions in SystemVerilog. Setting this variable to **true** enables the tool to use the name of the first union member as a reference for the port, net, and cell names associated with the union data type.

EXAMPLE

```
typedef union packed {
  logic R1;
  logic R2;
} betanode1;

typedef struct packed {
  logic [0:0]P1;
  betanode1 P2;
} subnode1;

typedef struct packed {
  betanode1 Q1;
} subnode2;

typedef union packed {
  subnode2 N2;
  logic  N1;
} node1;

typedef union packed {
  subnode1 M2;
  logic [1:0] M1;
} node2;

typedef struct packed {
 node2  A2;
 node1 [1:0] f1;
} packet;

module test (input packet p1, output packet p2, input clk);
```

```
  always @ (posedge clk)
    p2 = p1;
endmodule
```

When you set this variable to **false**, the following names are inferred:

```
Ports and nets: p1[A2][0], p1[A2][1], p1[f1][0][0], p1[f1][1][0]
Cells: p2_reg[A2][0], p2_reg[A2][1], p2_reg[f1][0][0], p2_reg[f1][1][0]
```

When you set this variable to **true**, the following names are inferred:

```
Ports and nets: p1[A2][M2][P1][0], p1[A2][M2][P2][R1],
 p1[f1][0][N2][Q1][R1], p1[f1][1][N2][Q1][R1]
Cells: p2_reg[A2][M2][P1][0], p2_reg[A2][M2][P2][R1],
 p2_reg[f1][0][N2][Q1][R1], p2_reg[f1][1][N2][Q1][R1]
```

SEE ALSO

hdlin_field_naming_style(3)
bus_dimension_separator_style(3)
hdlin_enable_upf_compatible_naming(3)

hdlin_sverilog_std

Specifies the standard that interprets SystemVerilog.

TYPE

string

DEFAULT

2012

DESCRIPTION

Use this variable to control the Formality RTL reader's default SystemVerilog language interpretation.

By default, the **read_sverilog** command interprets SystemVerilog read in the **2012** mode, for example, using IEEE Standard 1800-2012.

To change the default mode of **read_sverilog** compilation, enter **set hdlin_sverilog_std** "*value*", where *value* is one of the following:

- **2005** - Uses IEEE standard 1800-2005
- **2012** - Uses IEEE standard 1800-2012

The standard specified using the **read_sverilog** command overrides the value specified using the **hdlin_sverilog_std** variable.

EXAMPLE

The following example shows how to use the IEEE Standard 1800-2005 to interpret mydesign.v.

```
fm_shell > set hdlin_sverilog_std 2012
fm_shell > read_sverilog -05 -r mydesign.v
```

SEE ALSO

read_sverilog(2)

hdlin_systemverilog_default_automatic_tfs

Converts static functions/tasks in root/package/interface scope to automatic .

SYNTAX

set hdlin_systemverilog_default_automatic_tfs (true|false)

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

A static function is a function in which all declared items are statically allocated. These items shall be shared across all uses of the function executing concurrently. Functions defined within a module, interface, program, or package default to being static unless declared explicitly using the optional automatic keyword as part of the function declaration. An automatic function is reentrant, with all the function declarations allocated dynamically for each concurrent function call.

Formality does not support static tasks/functions in root/package/interface scope. Use this variable to convert all static functions to automatic internally. Example: function int count();//By default the function will considered as static unless specified otherwise (automatic) count++; return count; endfunction

module test(); always begin $display(count()); end endmodule

By setting this variable to true Formality will convert static functions/tasks in root/package/interface scope to automatic .

SEE ALSO

hdlin_tech_cell_seq_naming_in_to_out

Reverses the bit order of the bits of a synhronizer cell.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to reverse the bit order of the bits of a synchronizer cell. By default, Formality names the SEQ cells inside a synchronizer starting from its Q output pin. Each register appear in the path from Q pin to D pin of synchronizer will be assigned with a sequence number starting 0 to N. And the register will be named based on its sequence number like *dff.00*, *dff.01* to *dff.0N*.

If this variable is set, the registers will be named in reverse order i.e the register close to output will be named *dff.0N* and the register closed to 'D - input' will be named as *dff.00*.

To change the value of this variable, enter **set hdlin_tech_cell_seq_naming_in_to_out** "*value*", where *value* is "true" or "false".

* "true" - Reverse the order of the bits of synchronizer cell

* "false" - Don't reverse the order of the bits of synchronizer cell (default).

hdlin_unique_bbox_names

Controls the way Formality names black box designs

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

Formality issues an error if it cannot find a design to link with a specific cell. If User sets the variable "hdlin_unresolved_modules" to "black_box", then it generates black box designs in a library called "FM_BBOX" and then links the design to the cell.

By default, Formality generates blackbox design names using the reference design base-name, independent of cell parameters.

Set this variable to true to generate blackbox design names with the cell parameter information.

EXAMPLE

set hdlin_unique_bbox_names true

```
module top (output logic [1:0] out, input logic [1:0] in);
   mid #(.m(1), .n(0)) m1(out[1], in[1]);
endmodule
```

The blackbox design name generated for the cell m1 is : mid_m1_n0

SEE ALSO

hdlin_unresolved_modules(2)
hdlin_default_bbox_parameter_name(3)

hdlin_unresolved_modules

Specifies how to control black box creation for unresolved design references.

TYPE

string

DEFAULT

"error"

DESCRIPTION

Use this variable to control black box creation for Verilog or VHDL descriptions of references that are not resolved during set_top.

To change the value of this variable, enter **set hdlin_unresolved_modules** "*value*", where *value* is "black_box" or "error".

* "black_box" - Your unresolved Verilog and VHDL design references are turned into black boxes.

* "error" - Unresolved Verilog and VHDL design references are treated as errors.

SEE ALSO

set_top(2)

hdlin_upf_library

Designates that all cells within the specified libraries are UPF compliant.

TYPE

string

DEFAULT

DESCRIPTION

By default, the active (power on) state of the ground port in a IEEE-1801 standard (UPF) design is a logic 1. Libraries that follow UPF supply semantics are written so that a cell is in a power on state when the cell's power and ground pins are both at a logic 1 value. Most cell libraries currently in use do not follow those semantics, and use more traditional view that ground is active 0.

This variable is used to identify libraries that follow UPF supply semantics. It is only needed when you have these kinds of libraries and will be loading UPF as part of the design.

The value of this variable is a space separated list of library names. It must be set before issuing the set_top command, because it is used during set_top to correctly identify UPF compliant cells.

EXAMPLE

set hdlin_upf_library "stdlib floplib lslib"

SEE ALSO

set_top(2)

hdlin_upf_models

TYPE

string

DEFAULT

""

DESCRIPTION

The variable value is a string of model names. During set_top Formality will elaborate the models so that they will be available for use during load_upf. This variable is used in conjunction with the UPF construct map_retention_register -lib_model_name. Any model name that is specified in the UPF lib_model_name should be in the **hdlin_upf_models**.

This variable must be set before the set_top command occurs.

EXAMPLE

```
set hdlin_upf_models "generic_retention_flop generic_retention_latch"
set_top TOP
Setting top design to 'r:/WORK/TOP'
Status:   Elaborating design TOP   ...
Status:   Elaborating design generic_retention_flop   ...
Status:   Elaborating design generic_retention_latch   ...
Status:  Implementing inferred operators...
Top design successfully set to 'r:/WORK/TOP'
```

SEE ALSO

load_upf(2)

hdlin_use_hierarchical_register_names

Generates a hierachical register name.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable generate hierachical names for registers and signals in the following VHDL statements:

- VHDL GENERATE statement
- BLOCK statement
- PROCESS statement
- IF GENERATE statement

EXAMPLE

This example shows how to generate the following hierarchical names of the registers:

```
gl1(0)/named/old_reg
gl1(1)/named/old_reg
```

```
  gl1: for i in 0 to 1 generate
    named: process(clk)
      variable old : std_logic;
      begin
      if clk'event and clk = '1' then
       if( en = '1') then
       q(i) <= old;
      end if;
          old := data(i);
      end if;
    end process;
  end generate;
```

SEE ALSO

hdlin_hierarchy_separator_style(3)

hdlin_use_partial_modeled_cells

Specifies how Formality tests *misson mode only* components in the Synopsys .lib files and whether it treats them as black boxes.

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

This variable specifies how Formality tests *misson mode only* components defined in the Synopsys .lib files.

By default, Formality considers test components defined in Synopsys .lib files with the *misson mode only* functionality to be black boxes. Such components are defined with the "test_cell" construct in .lib source files.

To use the *misson mode* functionality of such cells, set the value of this variable to**true**. Note that the complete functionality of the design is not tested, as the test-mode functionality is not defined.

hdlin_use_vhdl_gen_hierarchy_for_naming

Generate hierachical names of instances.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable generates the hierachical names of instances name under the following VHDL statements:

- VHDL GENERATE statement
- BLOCK statement
- PROCESS statement
- IF GENERATE statement

EXAMPLE

This example generates the following hierarchical names of instances:

```
gl1(0)/u1
gl1(1)/u1

gl1: for i in 0 to 1 generate
u1 : etff port map(CLK => clk, ENABLE=> en, DATA => data(j*2+i), MUXDATA => q(j*2+i));
end generate;
```

SEE ALSO

hdlin_hierarchy_separator_style(3)

hdlin_v2005_replication_semantics

Controls tool behaviour for zero replication constant.

TYPE

string

DEFAULT

"true"

DESCRIPTION

The replication factor applied to a Verilog concatenation must be either positive (Verilog-2001 and earlier) or non-negative (Verilog-2005 and later).

Zero replication constant is made legal from Verilog-2005 (IEEE STD 1364-2005 and later). If the file was read using Verilog-2001 standard, the Formality tool considers the zero replication operation to have a size of zero and ignores it.

Set the value of this variable to false if you intend to generate a 1'b0 result for the zero replication constant.

To change the value of this variable, enter **set hdlin_v2005_replication_semantics** "*value*", where *value* is "true" or "false".

Note: This variable impacts only those designs read using the Verilog-2001 standard.

SEE ALSO

hdlin_vrlg_std(3)
read_verilog(2)

hdlin_verilog_directive_prefixes

Specifies a set of prefixes to be used for all the directives in Verilog files.

TYPE

string

DEFAULT

"synopsys formality pragma"

DESCRIPTION

Use this variable to specify a set of prefixes to be used for all the directives(sometimes called pragmas) in Verilog files. The value is a space separated list of strings.Each string is used to identify a directive comment in a Verilog file.

Change the value of this variable to tell Formality to try to process directives with prefixes that it does not normally recognize. Formality will issue a warning if it is not able to process a directive.Change this variable before using the **read_verilog** command to read the RTL with the specified directive prefixes.

Note:It will not apply to Verilog files that have previously been read.

DESCRIPTION

The following example enables Formality to recognize directives with prefixes of "pragma" and "synthesis" rather than the default prefixes.

```
fm_shell (setup)> set hdlin_verilog_directive_prefixes "pragma synthesis"
```

This will allow Formality to recognize Verilog RTL directives like this:

```
// pragma synthesis_off
$display("Error");
// pragma synthesis_on

case (state) // synthesis parallel_case
```

SEE ALSO

hdlin_ignore_synthesis(3)
hdlin_ignore_translate(3)
hdlin_vhdl_directive_prefixes(3)

hdlin_verilog_ignore_var_redeclaration

Allows Formality to ignore redeclaration of a variable

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

Formality by default throws an error if a variable is redeclared. Example:

module test(input a, output b); wire c; wire c; assign c = a; assign b = c;

endmodule

For the above code Formality throws an error for redeclaration of variable c. To avoid this error the variable hdlin_verilog_ignore_var_redeclaration can be set to true.

To change the value of this variable, use the following command: **set hdlin_verilog_ignore_var_redeclaration** "*value*" where *value* is "true" or "false".

SEE ALSO

read_verilog(2)
read_sverilog(2)

hdlin_verilog_named_generate_blocks

Controls the Formality Verilog default naming of unnamed generate blocks.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls the default names of generated blocks in Verilog LRM standards 2001. From Verilog LRM standard 2005 onwards Formality always generated default names for unnamed generate blocks. The Verilog LRM standards 1995 doesn't support the generate blocks.

By default, the **hdlin_verilog_named_generate_blocks** variable is set to false. By default, the tool doesn’t generate default names for unnamed generate blocks when read in for Verilog LRM standards 2001. To generate the default names for unnamed generate blocks set the variable to true.

Example:

module test(d,clk,rst,q); parameter N = 2; parameter P_CLONE = 0; input [N-1:0] d; input clk,rst; output reg [N-1:0] q;

generate //Unnamed generate block if(P_CLONE == 0) begin sub #(.N(N)) U1(.d(d),.clk(clk),.q(q),.rst(rst)); end endgenerate endmodule

module sub(d,clk,rst,q); parameter N = 2; parameter RST = 2'h0; input rst; input [N-1:0] d; input clk; output reg [N-1:0] q;

always @(posedge clk) begin if(rst) q <= RST; else q <= d; end endmodule

In above example for Verilog LRM standards 2001 Formality name the instance of 'sub' as "U1" by default. When 'hdlin_verilog_named_generate_blocks' set to true, Formality name it as "genblk1.U1" by adding 'genblk1' as a default name.

hdlin_verilog_reconstruct_multi_dimension_bus

Controls whether the Verilog netlist reader preserves multidimensional bus ports.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

When set to *true*, the multidimensional bus ports are preserved during read_verilog. When the variable is set to *false* (the default), multidimensional bus ports will be bit-blasted into individual ports.

EXAMPLE

This is an example of a multidimensional, structure port that has been bit blasted

module test (.din({\din.a[3] , \din.a[2] , \din.a[1] , \din.a[0] , \din.b , \din.c[3] , \din.c[2] , \din.c[1] , \din.c[0]}), .dout({\dout.a[3] , \dout.a[2] , \dout.a[1] , \dout.a[0] , \dout.b , \dout.c[3] , \dout.c[2] , \dout.c[1] , \dout.c[0]}));

input \din.a[3] , \din.a[2] , \din.a[1] , \din.a[0] , \din.b , \din.c[3] , \din.c[2] , \din.c[1] , \din.c[0] ; output \dout.a[3] , \dout.a[2] , \dout.a[1] , \dout.a[0] , \dout.b , \dout.c[3] , \dout.c[2] , \dout.c[1] , \dout.c[0] ;

SEE ALSO

read_verilog(2)

hdlin_verilog_wired_net_interpretation

Specifies whether non-wire nets are resolved using a simulation or a synthesis interpretation

TYPE

string

DEFAULT

"simulation"

DESCRIPTION

Synthesis resolves **wand** and **wor** nets by inserting logic gates (AND, OR). This produces a different result than simulation when a **Z** state could propagate to a compare point. Setting this variable to "simulation" (default) causes Formality to resolve the net according to simulation semantics. When the variable is set to "synthesis", Formality infers resolution logic during design read that is consistent with synthesis (prevents **Z** state propagation).

To change the value of this variable, enter **set hdlin_verilog_wired_net_interpretation** "*value*", where *value* may be "simulation" or "synthesis".

hdlin_vhdl_auto_file_order

Enables the read_vhdl command to automatically order the specified files.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to enable Formality to automatically order the files specified in the **read_vhdl** command. If set to "false" the files will be read in the order specified by the user.

To change the value of this variable, enter **set hdlin_vhdl_auto_file_order** "*value*", where *value* is "true" or "false".

SEE ALSO

read_vhdl(2)

hdlin_vhdl_directive_prefixes

Specifies a set of prefixes to be used for all the directives in VHDL files.

TYPE

string

DEFAULT

"synopsys formality pragma"

DESCRIPTION

Use this variable to specify a set of prefixes to be used for all the directives (sometimes called pragmas) in VHDL files. The value is a space separated list of strings. Each string is used to identify a directive comment in a VHDL file.

Change the value of this variable to tell Formality to try to process directives with prefixes that it does not normally recognize. Formality will issue a warning if it is not able to process a directive. Change this variable before using the **read_vhdl** command to read the RTL with the specified directive prefixes.

Note: It will not apply to VHDL files that have previously been read.

EXAMPLE

The following example enables Formality to recognize directives with prefixes of "pragma" rather than the default prefixes.

```
fm_shell (setup)> set hdlin_vhdl_directive_prefixes "pragma"
```

This will allow Formality to recognize VHDL RTL directives like this:

```
-- pragma synthesis_off
assert false "Error" severity Error
-- pragma synthesis_on
```

SEE ALSO

hdlin_ignore_synthesis(3)
hdlin_ignore_translate(3)
hdlin_verilog_directive_prefixes(3)

hdlin_vhdl_disable_file_reread

Controls analyzing of VHDL files with the same content that was analyzed previously.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The Formality tool by default analyzes each file that is read using the **read_vhdl** command, whether the file content is the same from the previously analyzed file or not.

To disable analyzing of multiple files with the same content more than once, set the **hdlin_vhdl_disable_file_reread** variable to **true**. So, multiple files with the same content read using the **read_vhdl** command are analyzed only one time.

This variable support is limited to files with packages of the same design library.

SEE ALSO

FMR_VHDL-528(n)
FMR_VHDL-202(n)

hdlin_vhdl_forgen_inst_naming

Specifies the scheme that should be used to name component instances within VHDL for-generate statements.

TYPE

string

DEFAULT

"mode0"

DESCRIPTION

Use this variable to specify the scheme Formality should use to name components instantiated inside for-generate statements.

To change the value of this variable, enter **set hdlin_vhdl_forgen_inst_naming** "*value*", where *value* is "mode0", "mode1", or "mode2".

* "mode0" - Appends the current value of the for-generate index to the user specified instance name separating the two parts with an underscore.

* "mode1" - Same as "mode0" expect when the component instantiated has a generic map. The first one instantiated gets the user specified instance name. The next one gets the user specified instance name with "_1" appended. Subsequent instances get "_2", "_3", and so forth appended.

* "mode2" - Same as "mode0" expect when the component instantiated has a generic map. The first one instantiated gets the user specified instance name. The next one gets the user specified instance name with "_2" appended. Subsequent instances get "_3", "_4", and so forth appended.

hdlin_vhdl_fsm_encoding

Specifies the FSM encoding for your VHDL description.

TYPE

string

DEFAULT

"binary"

DESCRIPTION

Use this variable to specify how FSMs are encoded from your VHDL description.

To change the value of this variable, enter **set hdlin_vhdl_fsm_encoding** "*value*", where *value* is "binary" or "1hot".

* "binary" - FSMs inferred from enumerated types will use a binary encoding. The resulting FSM will use the minimum number of FFs required to implement all FSM states.

* "1hot" - FSMs inferred from enumerated types will use a 1hot encoding. The resulting FSM will have one FF for each state. If 1hot encoding is specified, it may still be necessary to constrain the FSM as 1hot (using the set_constraint command) to achieve a passing verification.

SEE ALSO

set_constraint(2)

hdlin_vhdl_integer_range_constraint

Controls whether Formality adds logic to constrain the value of integer ports and registers to match the integer range constraint specified in the VHDL.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Setting this variable (true) causes Formality to introduce additional logic into the design to make sure that values coming from the integer ports, signals, and variables are constrained to the range specified in the source VHDL. This variable affects only primary inputs, registered signals and registered variables. Formality treats all values outside of the integer range constraint as don't care. Enabling this variable might allow a verification to succeed if the failures are caused by values outside of the constrained integer range.

For example,

entity test is port(int1 :in integer range 0 to 19); end test;

When synthesized, this input port will require 5 bits, allowing for possible values from 0 to 31. When this variable is true, Formality constrains the value on that port such that all values outside of the range 0 to 19 will be don't care.

For a list of **hdl** variables and their current values, type **printvar hdlin***.

SEE ALSO

hdlin_dyn_array_bnd_check(3)

hdlin_vhdl_mixed_language_instantiation

Controls the support for direct instantiation of Verilog or SystemVerilog Modules in VHDL without the use of a component or entity declaration.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable supports the implementation of the mixed language instantiations using direct entity instantiation. When the variable is set to **true**, it instantiates a Verilog or a SystemVerilog module in a VHDL architecture.

- During the direct instantiation, use of a literal is not allowed in a port and generic map.
- When an instance binding specifies both the entity and architecture, the specified architecture is slected. In case the architecture is not specified, the current design is elaborated.
- When the VHDL entity is not present and the port type in the instance and module does not match, the tool reports an error.
- When an entity does not have an architecture in the specified library and if the specified library has a module with the same name than the instance is linked to the module.

hdlin_vhdl_others_covers_extra_states

Enables the use a redundant others clause in a case statement to cover undefined states

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to control if others clauses are used to determine how unreachable states are handled.

When the number of elements in a VHDL enumerated type is not an exact power of two, the extra states generated by Formality are considered unreachable and therefore their behavior is undefined by simulation. When this variable to true, in a VHDL case statement which switches on an enumerated type (or constrained INTEGER) and has an others clause, the others clause is used is used to determine the behavior for the unreachable states. Setting this variable to false causes signals and variables assigned in the case statement to take on a don't care value in the unreachable states. This is to match the way some synthesis tools optimize the unreachable states.

To change the value of this variable, enter **set hdlin_vhdl_others_covers_extra_states** "*value*", where *value* is "true" or "false".

For example: type three_state is (IDLE, START, FINISH);

process(state) case state is when IDLE => next_state <= START; error <= '0'; when START => next_state <= FINISH: when FINISH => next_state <= IDLE; error <= '0'; when OTHERS => next_state <= IDLE; error <= '1'; end case end process;

In this example, the OTHERS clause cannot be reached in simulation. When synthesized, there are two state bits representing the signal state (type three_state), allowing four possible states for only three elements in the enumerated type three_state. When this variable is true, the fourth unreachable state is considered to be covered by the others clause and the assignments in the others clause will be used. When this variable is set to false, the assignments in the others clause will be ignored and any signals assigned in the case statement (next_state and error in this example) will be assigned a don't care value in the unreachable state.

hdlin_vhdl_presto_naming

Controls the generation of operator names inferred from VHDL.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to control whether inferred operator names generated by Formality are compatible with names produced by the Presto or names produced by HDLC. If set to "false" HDLC style operator names are generated. If set to "true" Presto style operator names are generated.

To change the value of this variable, enter **set hdlin_vhdl_presto_naming** "*value*", where *value* is "true" or "false".

hdlin_vhdl_presto_shift_div

Controls the implementation of a divide by a constant power of two.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to control how an inferred divide by a constant power of two is handled in Formality.

If you set this variable to "true", Formality creates a simulation compatible divider when the numerator of the divide (i.e. the dividend) is a signed value and the divisor is a constant power of two. This divider is consistent with the divider created by the Design Compiler Presto VHDL reader.

If you set this variable to "false", Formality always generates a simple shift when the divisor is a constant power of two. It will also produce a simulation/synthesis mismatch message when this operator is used with a dividend that signed, indicating the discrepancy this causes. This divider and message provide compatibility with the Design Compiler HDLC VHDL reader.

For a list of **hdl** variables and their current values, type **printvar hdlin***.

SEE ALSO

hdlin_vhdl_presto_naming(3)

hdlin_vhdl_std

Controls whether the Formality VHDL reader uses the VHDL 1987 standard, the 1993 standard, or the 2008 standard.

TYPE

string

DEFAULT

2008

DESCRIPTION

This variable determines the standard that the Formality's Vhdl Reader uses.

Specify one of the following values:

- **1993** - to use the VHDL 1993 standard.
- **1987** - to use the VHDL 1987 standard
- **2008** - to use the VHDL 2008 standard. This is the default.

Note that VHDL standard specified using the **read_vhdl** command override the value of the **hdlin_vhdl_std** variable.

SEE ALSO

read_vhdl(2)

hdlin_vhdl_strict_libs

Controls whether a strict mapping of VHDL libraries will be used during analysis.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Controls whether a strict mapping of VHDL libraries will be used during analysis. The default is FALSE. Having this variable disabled (false), allows Formality to be flexible in interpreting use clauses. If the design unit is not found in the specified library it will check the WORK library to see if its defined there and use it if found.

Note: This variable should be set before reading any VHDL files into Formality.

For a list of **hdl** variables and their current values, type **printvar hdlin***.

hdlin_vhdl_use_87_concat

Controls whether the IEEE Std 1076-1987 style concatenations are used in IEEE Std 1076-1993 VHDL code.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Enabling this variable (true) causes VHDL 1987 style concatenation indexing to be used in VHDL 1993 code. This variable enables compatibility with older versions of Formality and Design Compiler that accepted the code.

This variable has no effect when hdlin_vhdl_87 has the value true. When hdlin_vhdl_87 is true Formality will use IEEE Std 1076-1987 style concatenation indexing only.

When Formality issues an FMR_ELAB-381 message (Error: Range mismatch - null range on LHS and not on RHS.) during set_top it is possible that the out of range condition is caused by this inconsistent mixing of the VHDL 1987 and 1993 standards. Change the value of this variable to true, then read and set_top again.

The main cause of this problem is indexing into the result of a concatenation, which can be different in IEEE Std 1076-1993 and IEEE Std 1076-1987. For example, the following architecture will not set_top successfully unless you set hdlin_vhdl_use_87_concat to true.

ARCHITECTURE rtl OF test IS type bitvec is array (integer range <>) of bit; signal in1 : bitvec(15 DOWNTO 0); signal out1 : bitvec(20 DOWNTO 0);

FUNCTION zext (a : bitvec; size : INTEGER := 1) RETURN bitvec IS VARIABLE result : bitvec(a'HIGH+size DOWNTO a'LOW); BEGIN result := (OTHERS => '0'); result(a'RANGE) := a; RETURN result; END zext;

BEGIN out1 <= zext(in1 & '0', 4); END rtl;

For a list of **hdl** variables and their current values, type **printvar hdlin***.

SEE ALSO

hdlin_vhdl_std(3)
FMR_ELAB-381(n)

hdlin_vrlg_std

Controls the Formality Verilog RTL reader's default Verilog language interpretation.

TYPE

String

DEFAULT

2005

DESCRIPTION

Use this variable to control the Formality RTL reader's default Verilog language interpretation.

By default, the read_verilog command will interpret Verilog source code read in mode "2005", e.g., using IEEE standard 1364-2005.

To change the default mode of read_verilog compilation, enter **set hdlin_vrlg_std** "*value*", where *value* may be any of 1995, 2001 or 2005.

* "1995" - Uses IEEE standard 1364-1995

* "2001" - Uses IEEE standard 1364-2001

* "2005" - Uses IEEE standard 1364-2005

Note that if the read_verilog commandline contains a switch to control the Verilog dialect, then that value will override the value of the variable hdlin_vrlg_std.

For example, given: set hdlin_vrlg_std 1995 read_verilog -01 -r foo.v

The Formality Verilog reader will interpret file foo.v in IEEE standard 1364-2001 mode.

SEE ALSO

read_verilog(2)

hdlin_while_loop_iterations

Places an upper bound on the number of times a loop is unrolled (to prevent potential infinite loops).

TYPE

string

DEFAULT

4096

DESCRIPTION

Places an upper bound on the number of times a loop is unrolled (to prevent potential infinite loops). Loop unrolling occurs until the loop terminates. If you know that your loop will execute more times than the limit allows and that your loop will terminate at some point, increase the value of this variable.

hdlin_xlrm_resolve_overloaded_functions

Generates the FMR_VHDL-048 error for the overloaded function of IEEE std_logic_1164 package.

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

The IEEE std_logic_1164 has overloaded functions for BIT_VECTOR and STD_LOGIC_VECTOR. If a bit literal is passed as an argument to a function call, as the bit literal can be treated as both bit_vector and std_logic_vector, the tool reports the VHDL_ELAB-047error as per VHDL LRM. The value of bit_literal can have only known values that expand to strings of 0 or 1. The two functions from the 1164 package will always return identical values for a bit_literal argument. In such case this variable when set to true will enable the user to have a successful set top.

EXAMPLE

constant abc : STD_ULOGIC_VECTOR(31 downto 0) := to_stdulogicvector(X"00000000");

function "to_stdulogicvector" matches functions:

function TO_STDULOGICVECTOR (S : STD_LOGIC_VECTOR) return STD_ULOGIC_VECTOR function TO_STDULOGICVECTOR (B : BIT_VECTOR) return STD_ULOGIC_VECTOR

SEE ALSO

FMR_VHDL-047(n)

impl

Indicates the current implementation design.

TYPE

string

DEFAULT

DESCRIPTION

This variable is a read-only variable set by the application to indicate the current implementation design.

SEE ALSO

ref(3)

library_assume_pg_pins

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

This variable controls whether Formality creates one default power and one default ground PG pins for cells in non-PG pin libraries.

SEE ALSO

library_pg_file_pattern(3)

library_interface_only

Enables creation of a library cell(s) or technology design(s) consisting only of an interface.

TYPE

String

DEFAULT

DESCRIPTION

Use this variable to direct Formality's Verilog, VHDL and DB readers to produce interface-only library cells or technology designs. Resulting designs consist only of the source design's specified port declarations. The remaining design internals are either not synthesized (in the case of Verilog or VHDL behavioral RTL) or discarded (for Verilog netlist and DB based designs).

Formality design readers compare the name of the design it is currently processing to the value of this variable. If a match is found, the design is converted to an interface-only design. Compare rules are based on TCL's glob-style pattern matching syntax.

To change the value of this variable,

set library_interface_only "*value*"

where *value* is in TCL glob-style pattern matching form.

SEE ALSO

hdlin_interface_only(3)

library_pg_file_pattern

TYPE

String

DEFAULT

""

DESCRIPTION

This variable is used to locate the PG Tcl side file for library PG conversion. By default, the variable is set to "", which means that there is no PG Tcl file unless specified.

String substituion can also be used for finding PG Tcl side file: __DIR__ : Path to .db directory __FILE__: Leaf file name for .db

There can be one Tcl file for all DBs, one Tcl file per DB, or one Tcl file for a group of DBs.

Examples:

1. One Tcl for all DBs:

a. At the current directory set library_pg_file_pattern "libpg_sidefile.pg"

b. At different location set library_pg_file_pattern "/my_dir/libpg_sidefile.pg"

2. One Tcl per DB:

a. At the same location as original DB files set library_pg_file_pattern "__DIR__/__FILE__.pg"

b. At some directory called "pg_sidefiles" under the same dir as original DB set library_pg_file_pattern "__DIR__/pg_sidefiles/__FILE__.pg"

c. At a different location called "my_dir" set library_pg_file_pattern "/my_dir/__DIR__/__FILE__.pg"

3. One Tcl for a group of DBs:

a. At the same location as DB files set library_pg_file_pattern "__DIR__/libpg_sidefile.pg"

This variable must be set before loading the library. Otherwise, it will be ignored.

SEE ALSO

library_assume_pg_pins(3)

link_allow_physical_variant_cells

Allows the Formality tool to link physical variant cells to their master cell in a .db technology library.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

When the link_allow_physical_variant_cells variable is set to true, during **set_top** the Formality tool will link instances of physical variant cells in a netlist to the corresponding master cell in the .db technology library. See the formality.log file for messages about which physical variant cells were linked to master cells in each design.

message_level_mode

Sets the message severity threshold that Formality uses during verification.

TYPE

string

DEFAULT

"info"

DESCRIPTION

Use this variable to set the message severity threshold that Formality uses during verification.

Formality issues four types of messages; three of which leave the tool running, and one (fatal) that causes it to exit. The three types of nonfatal messages are errors, warnings, and informational notes.

* Errors occur when Formality cannot continue processing the current command.

* Warnings occur when Formality encounters a possible error situation.

* Notes are informational text alerting you to verification progress.

By default, Formality issues all three nonfatal messages during verification. However, you can establish a lower-level severity message threshold by using the **message_level_mode** variable. Setting the threshold to a particular level causes Formality to limit messages to that level and higher.

To change the value of this variable, enter **set message_level_mode** "*value*", where "*value*" is one of the following: "error", "warning", or "info".

* "error" - Causes Formality to return error-level messages only. With this setting, Formality does not return warning or informational messages.

* "warning" - Causes Formality to return warning and error-level messages only. With this setting, Formality does not return informational messages.

* "info" - Causes Formality to return informational, warning, and error-level messages (all types of Formality messages). This is the default setting.

message_x_source_reporting

Enables reporting of X sources in the design.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

Use this variable to enable X source reporting. Reporting is performed during matching and details are printed to formality.log file.

Following are considered as X sources:

* a DC cell output whose DC input is not constant0

* Un-driven nets/pins (depending on the setting of TCL variable verification_set_undriven_signals)

* constant-X SEQs

* Multiply driven, non-constant nets

To change the value of this variable, enter **set message_x_source_reporting** "*value*", where "*value*" is one of the following: "true" or "false".

SEE ALSO

verification_set_undriven_signals(3)

mw_logic0_net

Specifies the name of the Milkyway ground net.

TYPE

string

DEFAULT

"VSS"

DESCRIPTION

Use this variable to specify the name of the Milkyway ground net. The default name is "VSS".

To change the value of this variable, enter **set mw_logic0_net** "*name*".

SEE ALSO

mw_logic1_net(3)

mw_logic1_net

Specifies the name of the Milkyway power net.

TYPE

string

DEFAULT

"VDD"

DESCRIPTION

Use this variable to specify the name of the Milkyway power net. The default name is "VDD".

To change the value of this variable, enter **set mw_logic1_net** "*name*".

SEE ALSO

mw_logic0_net(3)

name_match

Specifies whether compare point matching uses object names, or relies solely on function and topology to match compare points.

TYPE

string

DEFAULT

"all"

DESCRIPTION

Use this variable to control whether compare point matching uses object names, or relies solely on function and topology to match compare points.

To change the value of this variable, enter **set name_match** "*value*", where *value* may be "all", "none", "port", or "cell".

* "all" - enables all name-based matching (the default)

* "none" - enables name-based matching of primary input ports, but not primary outputs, registers or black-box inputs

* "port" - enables name-based matching of all primary ports, but not registers or black-box inputs

* "cell" - enables name-based matching of registers and other cells, but not primary ports

name_match_allow_subset_match

Specifies whether and which name subset(token)-based name matching methods to use.

TYPE

string

DEFAULT

"strict"

DESCRIPTION

Use this variable to specify whether and which name subset(token)-based name matching methods to use: "strict", "any", or "none". A token is a sequence of all alphabetic or all numeric characters, delimited by filter characters. In the name abc/d1/e[*2*] the tokens are abc, d, 1, e, 2.

Standard default filter-based matching ignores delimiter characters, as long as doing so does not create name collisions, for example, a/b/c matches a~b~!@#c.

"Strict" subset matching further ignores tokens that appear in at least 90% of all names of a given type of object, as long as doing so does not create name collisions, for example, given

data[0] data[1] data[2] data[3] data[4] bus[0] bus[1] bus[2] bus[3] bus[4]

versus

data_reg[0] data_reg[1] data_reg[2] data_reg[3] data_reg[4] bus_reg[0] bus_reg[1] bus_reg[2] bus_reg[3] bus_reg[4]

"strict" subset matching will ignore the token "reg", matching data[*] to data_reg[*] and bus[*] to bus_reg[*].

"Any" subset matching tries to choose the best match among multiple candidates. It ignores the same tokens that "strict" matching ignores. A match is a candidate if there is at least one matching token between the two names, and either all of the unignored tokens in one of the names is matched, or at least 2/3 of the unignored tokens in both names are matched, or at least 1/2 of the unignored tokens in both names are matched and there is no unmatched numeric token in either name. Which candidate is considered best depends on the number of matched vs unmatched tokens, and whether any numeric tokens are unmatched, and if they are where they appear in the name.

If subset matching is "strict", it is applied before signature analysis.

If subset matching is "any", then "strict" subset matching is applied before signature analysis and "any" subset matching is applied to points remaining unmatched after signature analysis.

To change the value of this variable, enter **set name_match_allow_subset_match** "*value*", where *value* may be "strict", "any", or "none".

SEE ALSO

name_match_use_filter(3)
name_match_filter_chars(3)
signature_analysis_allow_subset_match(3)

name_match_based_on_nets

Specifies whether compare points are matched based on attached net names.

TYPE

string

DEFAULT

"auto"

DESCRIPTION

Use this variable to control the matching of compare points based on attached net names. When the names of compare points do not match, the Formality tool attempts to match compare points based on the names of the attached nets.

The allowed values of this variable are as follows:

* auto: Turns off net-name based matching when SVF is set. If SVF is not set, the tool allowsnet-name based compare point matching.

* false: Turns off net-name based compare point matching

* true: Allows net-name based compare point matching

To change the value of this variable, enter **set name_match_based_on_nets** "*value*" at the tool prompt, where the *value* can be "auto", "true" or "false".

SEE ALSO

name_match_use_filter(3)
name_match_filter_chars(3)
name_match_allow_subset_match(3)
name_match(3)

name_match_filter_chars

Specifies the characters that should be ignored when the name matching filter is used.

TYPE

string

DEFAULT

"'~!@#$%^&*()_-+=|\[]{}""

DESCRIPTION

Use this variable to specify the characters that should be ignored when the name matching filter is used.

To change the value of this variable, enter **set name_match_filter_chars** "*value*", where "*value*" may be a list of characters enclosed with braces, such as "{/>*(}". This will cause the characters "/", ">", "*", and "(" to be ignored in all names.

Here are some examples:

```
fm_shell> set name_match_filter_chars \
"{$%^&*()_-+} $%^&*()_-+"

fm_shell> set name_match_filter_chars \
"{`~!@#$%^&* ()_-+=|\[]{}"':;?,./}`~!@#$%^&*\
()_-+=|\[]{}"':;?,./"

fm_shell> set name_match_filter_chars \
"{`~!@#$%^&*()_-+=|\[]{}"':;?,./ab}\
`~!@#$%^&*()_-+=|\[]{}"':;?,./ab"
```

Consider the last example; within the braces we have all the characters from the previous value and the characters "ab". Another way, for this same effect, is by using Tcl commands to append the additional characters you wish to add to the current value of the variable. The following examples illustrate how:

```
fm_shell> append name_match_filter_chars \
"ab" "`~!@#$%^&*()_-+=|\[]{}"':;?,./ab"

fm_shell> exp = cdefgh
Information: Defining new variable "exp". (CMD-041)
cdefgh

fm_shell> append name_match_filter_chars \
$exp "`~!@#$%^&*()_-+=|\[]{}"':;?,./abcdefgh"
```

name_match_flattened_hierarchy_separator_style

Specifies the separator used in pathnames Formality creates when it flattens a design during hierarchical verification.

TYPE

string

DEFAULT

"/"

DESCRIPTION

You can use this variable to specify the character Formality uses when creating flattened pathnames during hierarchical verification. During hierarchical verification, Formality flattens hierarchical blocks that fail verification in their isolated context. In doing so, Formality needs to create fully-flattened pathnames that represent design objects. The default character Formality uses as a separating character is the backslash "/". You can redefine the separator by supplying any single character as the separator style. For information about flattening of designs during hierarchical verification, refer to "Hierarchical Verification" in the user guide.

To change the value of this variable, enter **set name_match_flattened_hierarchy_separator_style** "*value*", where *value* should be one character only.

name_match_multibit_register_reverse_order

Reverses the bit order of the bits of a multibit register.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to reverse the bit order of the bits of a multibit register. By default, Formality will automatically match multibit registers to their corresponding single bit counterparts based on their name and bit order. If the bit order has been changed after synthesis, you must set this variable to "true".

To change the value of this variable, enter **set name_match_multibit_register_reverse_order** "*value*", where *value* is "true" or "false".

* "true" - Reverse the order of the bits of multibit registers.

* "false" - Don't reverse the order of the bits of multibit registers (default).

name_match_net

Specifies whether name matching attempts to match reference nets to implementation nets.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to control whether name matching attempts to match previously unmatched reference nets to implementation nets. Such matches identify potential cut-points and may improve verification performance.

By default, name matching does not attempt match previously unmatched reference nets to implementation nets. To force name matching to attempt to match previously unmatched reference nets to implementation nets, set this variable to "true".

To change the value of this variable, enter **set name_match_net** *value*, where *value* is "true" or "false".

SEE ALSO

name_match(3)
name_match_based_on_nets(3)
name_match_filter_chars(3)
name_match_pin_net(3)
name_match_use_filter(3)
signature_analysis_match_net(3)

name_match_pin_net

Specifies whether name matching attempts to match hierarchical pins to nets.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to control whether name matching attempts to match previously unmatched hierarchical pins in one design to nets in the other design. Such matches identify potential cut-points and may improve verification performance.

By default, name matching does not attempt to match previously unmatched hierarchical pins to nets. To force name matching to attempt to match previously unmatched hierarchical pins to nets, set this variable to "true".

To change the value of this variable, enter **set name_match_pin_net** *value*, where *value* is "true" or "false".

SEE ALSO

name_match(3)
name_match_based_on_nets(3)
name_match_filter_chars(3)
name_match_net(3)
name_match_use_filter(3)
signature_analysis_match_pin_net(3)

name_match_use_filter

Specifies whether the built-in name matching filter should be used.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to specify whether the built-in name matching filter should be used. Here are the filter rules for matching:

1. All characters in the ignore list are replaced by an "_".

2. If the ignored character is the first or the last character, then it is not replaced by "_", but discarded completely.

3. Digits are separated from characters with a "_". For example, "bar2" is translated to "bar_2".

4. If two strings in the same design are translated or normalized to the same string, then neither string is matched by name matching.

For information on ignored characters, refer to the man page on **name_match_filter_chars**.

To change the value of this variable, enter **set name_match_use_filter** "*value*", where *value* is "true" or "false".

SEE ALSO

name_match_filter_chars(3)

orig_impl

Indicates the current original implementation design.

TYPE

string

DEFAULT

DESCRIPTION

This variable is a read-only variable set by the application to indicate the current original implementation design specified with the set_orig_implementation command.

SEE ALSO

orig_ref(3)
eco_ref(3)
eco_impl(3)

orig_ref

Indicates the current original reference design.

TYPE

string

DEFAULT

DESCRIPTION

This variable is a read-only variable set by the application to indicate the current original reference design specified with the set_original_reference command.

SEE ALSO

orig_impl(3)
eco_ref(3)
eco_impl(3)

port_complement_naming_style

Defines the convention that synthesis uses to rename ports that were complemented.

TYPE

string

DEFAULT

"%s_BAR"

ENABLED SHELL MODES

setup

DESCRIPTION

This variable defines the convention that synthesis uses to rename ports that were complemented. The variable string must either be the empty string or contain one occurrence of %s (percent s).

Formality will automatically match with inverted polarity a reference design port with the original name to an implementation design port that conforms to the specified naming style. For example with the default setting, reference port "foo" will be matched to implementation port "foo_BAR" with inverted polarity.

Setting the variable to the empty string will disable this behavior.

SEE ALSO

variable(3)

ref

Indicates the current reference design.

TYPE

string

DEFAULT

DESCRIPTION

This variable is a read-only variable set by the application to indicate the current reference design.

SEE ALSO

impl(3)

save_session_calculate_cone_sizes

This variable specifies the types of compare points for which Formality calculates cone sizes when saving a session.

TYPE

String

DEFAULT

""

DESCRIPTION

When this variable is set to any combination of the valid compare point verification status values, the tool calculates cone sizes for all compare points of specified status during the save_session command. When the session is restored, the user will not need to explicitly calculate those sizes in the GUI.

The possible compare point verification status values:

- **failing** - compare points (CPs) failing verification
- **aborted** - CPs deemed inconclusive during verification
- **passing** - CPs that passed verification
- **unverified** - CPs excluded from verification run

By default, the value is an empty string and no cone sizes are calculated.

EXAMPLES

To calculate the size of failing and aborted compare points during save_session

```
fm_shell> set save_session_calculate_cone_sizes "failing aborted"
```

SEE ALSO

save_session(2)

schematic_cone_collapse_disabled_dont_care

Specifies how the schematic view of the logic cone displays a Dont Care cell which can never assert dont care because of a constant on its control pin.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When this variable is set to *true* (default), any Dont Care cell in the cone that has a constant value of '0' on its DC pin will be replaced with a small square 'x' symbol that represents the disabled Dont Care cell and its buffer function. This will reduce the complexity of the cone schematic.

When the variable is set to *false*, the Dont Care cells will all be displayed full size with both input pins and their connections.

To determine the value of this variable, enter **printvar schematic_cone_collapse_disabled_dont_carefP".**

schematic_cone_collapse_supply_network

Specifies how the schematic view of the logic cone displays the supply network in ALL_ON mode.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When this variable is set to *true*, and the verification is in **ALL-ON** mode, the cone will stop and exclude any switches/macros driving the supply. This will reduce the complexity of the cone schematic.

When the variable is set to *false*, the full supply network will be included in the cone unless a cutpoint has been set.

To determine the value of this variable, enter **printvar schematic_cone_collapse_supply_networkfP".**

schematic_expand_logic_cone

Specifies whether the schematic view of the logic cone displays the internals of techlib cells and DesignWare components.

TYPE

string

DEFAULT

"auto"

DESCRIPTION

Use this variable to control whether the internals of techlib cells and DesignWare components are displayed in subsequently generated logic cones.

The schematic view of the logic cone normally hides the internals of most techlib cells and DesignWare components to reduce clutter and make the schematics easier to read. By default, techlib cells containing multiple sequential elements, feedback loops or additional logic that could modify the output value are automatically expanded by default.

To change the value of this variable, enter **set schematic_expand_logic_cone** "*value*", where *value* is "true", "auto" or "false".

Setting the value to "false" will prevent techlib cells from being expanded. Note that not expanding the cells can cause duplicate entries and conflicting logic values to be displayed in the pattern view.

Setting the value to "true" will cause all techlib cells to be expanded. Note that only the internal elements that are part of the cone of logic are displayed.

To determine the value of this variable, enter **printvar schematic_expand_logic_cone**".

search_path

Specifies directories searched for design and library files specified without directory names.

TYPE

string

DEFAULT

" "

DESCRIPTION

Use this variable to specify directories searched for design and library files specified without directory names. This variable is a list of directory names and is usually set to a central library directory. Commands like **read_db** depend heavily on **search_path**.

It is possible to get the **source** command to search for scripts using **search_path** by setting the **sh_source_uses_search_path** variable to "true".

To change the value of this variable, enter **set search_path** "*value*", where *value* is a list of space separated directory names. By surrounding the path list with double quotation marks, the environment variables listed as part of a path are expanded.

SEE ALSO

sh_source_uses_search_path(3)
source(2)

sh_allow_tcl_with_set_app_var

Allows the **set_app_var** and **get_app_var** commands to work with application variables.

TYPE

string

DEFAULT

application specific

DESCRIPTION

Normally the **get_app_var** and **set_app_var** commands only work for variables that have been registered as application variables. Setting this variable to **true** allows these commands to set a Tcl global variable instead.

These commands issue a CMD-104 error message for the Tcl global variable, unless the variable name is included in the list specified by the **sh_allow_tcl_with_set_app_var_no_message_list** variable.

SEE ALSO

get_app_var(2)
set_app_var(2)
sh_allow_tcl_with_set_app_var_no_message_list(2)

sh_allow_tcl_with_set_app_var_no_message_list

Suppresses CMD-104 messages for variables in this list.

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable is consulted before printing the CMD-104 error message, if the **sh_allow_tcl_with_set_app_var** variable is set to **true**. All variables in this Tcl list receive no message.

SEE ALSO

get_app_var(2)
set_app_var(2)
sh_allow_tcl_with_set_app_var(2)

sh_arch

Indicates the current system architecture of the machine you are using.

TYPE

string

DEFAULT

"sparcOS5"

DESCRIPTION

This variable is a read-only variable set by the application to indicate the current system architecture of the machine you are using such as sparc, hpux, sparcOS5, and so on. The **sh_arch** variable is a read-only variable.

sh_command_abbrev_mode

Sets the command abbreviation mode for interactive convenience.

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable sets the command abbreviation mode as an interactive convenience. Script files should not use any command or option abbreviation, because these files are then susceptible to command changes in subsequent versions of the application.

Although the default value is **Anywhere**, it is recommended that the site startup file for the application set this variable to **Command-Line-Only**. It is also possible to set the value to **None**, which disables abbreviations altogether.

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

SEE ALSO

sh_command_abbrev_options(3)
get_app_var(2)
set_app_var(2)

sh_command_abbrev_options

Turns off abbreviation of command dash option names when false.

TYPE

boolean

DEFAULT

application specific

DESCRIPTION

When command abbreviation is currently off (see sh_command_abbrev_mode) then setting this variable to false will also not allow abbreviation of command dash options. This variable also impacts abbreviation of the values specified to command options that expect values to be one of an allowed list of values.

This variable exists to be backward compatible with previous tool releases which always allowed abbreviation of command dash options and option values regardless of the command abbreviation mode.

It is recommended to set the value of this variable to false.

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

SEE ALSO

sh_command_abbrev_mode(3)
get_app_var(2)
set_app_var(2)

sh_command_log_file

Specifies the name of the file to which the application logs the commands you executed during the session.

TYPE

string

DEFAULT

empty string

DESCRIPTION

This variable specifies the name of the file to which the application logs the commands you run during the session. By default, the variable is set to an empty string, indicating that the application's default command log file name is to be be used. If a file named by the default command log file name cannot be opened (for example, if it has been set to read only access), then no logging occurs during the session.

This variable can be set at any time. If the value for the log file name is invalid, the variable is not set, and the current log file persists.

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

SEE ALSO

get_app_var(2)
set_app_var(2)

sh_continue_on_error

Allows script processing to continue when errors occur.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to determine whether script processing can continue when errors occur. Under normal circumstances, when executing a script with source, an error causes the script processing to terminate.

Setting **sh_continue_on_error** to "true" allows processing to continue when errors occur. By default, this variable is set to false.

To change the value of this variable, enter **set sh_continue_on_error** "*value*", where *value* is "false" or "true".

sh_deprecated_is_error

Raise a Tcl error when a deprecated command is executed.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When set this variable causes a Tcl error to be raised when an deprecated command is executed. Normally only a warning message is issued.

SEE ALSO

get_app_var(2)
set_app_var(2)

sh_dev_null

Indicates the current null device.

TYPE

string

DEFAULT

platform dependent

DESCRIPTION

This variable is set by the application to indicate the current null device. For example, on UNIX machines, the variable is set to /**dev**/**null**. This variable is read-only.

SEE ALSO

get_app_var(2)

sh_enable_line_editing

Enables the command line editing capabilities in Formality.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

If set to "true" it enables advanced unix like shell capabilities.

This variable needs to be set in .synopsys_fm.setup file to take effect.

Key Bindings

The **list_key_bindings** command displays current key bindings and the edit mode. To change the edit mode, variable **sh_line_editing_mode** can be set in either the .synopsys_fm.setup file or directly in the shell.

Command Completion

The editor will be able to complete commands, options, variables and files given a unique abbreviation. User need to type part of a word and hit the tab key to get the complete command, variable or file. For command options, users need to type '-' and hit tab key to get the options list.

If no match is found, the terminal bell rings. If the word is already complete a space is added to the end, if it isn't already there, to speed typing and provide a visual indicator of successful completion. Completed text pushes the rest of the line to the right. If there are multiple matches then all the matching commands/options/files or variables are autolisted.

Completion works in following context sensitive way :-

The first token of a command line : completes commands

Token that begins with "-" after a command : completes command arguments

After a ">", "|" or a "sh" command : completes filenames

After a set, unset or printvar command : completes the variables

After '$' symbol : completes the variables

After the help command : completes command

After the man command : completes commands or variables

Any token which is not the first token and doesn't match any of the above rules : completes filenames

SEE ALSO

sh_line_editing_mode(3)
list_key_bindings(2)

sh_enable_page_mode

Specifies how long reports are displayed.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to indicate how long reports display.

When "true", long reports are displayed one page at a time (similar to the UNIX **more** command).

To change the value of this variable, enter **set sh_enable_page_mode** *value*, where *value* is "false" or "true".

sh_enable_stdout_redirect

Allows the redirect command to capture output to the Tcl stdout channel.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When set to **true**, this variable allows the redirect command to capture output sent to the Tcl stdout channel. By default, the Tcl **puts** command sends its output to the stdout channel.

SEE ALSO

get_app_var(2)
set_app_var(2)

sh_help_shows_group_overview

Changes the behavior of the "help" command.

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable changes the behavior of the **help** command when no arguments are specified to help. Normally when no arguments are specified an informational message with a list of available command groups is displayed.

When this variable is set to false the command groups and the commands in each group is printed instead. This variable exists for backward compatibility.

SEE ALSO

help(2)
set_app_var(2)

sh_line_editing_mode

Enables vi or emacs editing mode in Formality shell.

TYPE

String

DEFAULT

"emacs"

DESCRIPTION

This variable can be used to set the command line editor mode to either "vi" or "emacs". Valid values are "emacs" or "vi".

Use **list_key_bindings** command to display the current key bindings and edit mode.

This variable can be set in the either .synopsys_fm.setup file or directly in the shell. The **sh_enable_line_editing** variable must be set to "true".

SEE ALSO

sh_enable_line_editing(3)
list_key_bindings(2)

sh_man_browser_mode

Controls whether man pages are displayed in a Web browser window or in the shell transcript window.

TYPE

string

DEFAULT

"gui"

DESCRIPTION

This variable specifies when the man pages are displayed in a Web browser or in the shell transcript window.

Specify one of the following values

- **gui** - to display the man page in a Web browser window when the GUI is open.
- **shell** - to display the man page in a Web browser window when the GUI is closed.
- **both** - to always display the man page Web browser window irrespective of whether the GUI is open or closed.
- **none** - to display the man page in the shell transcript window irrespective of whether the GUI is open or closed.

SEE ALSO

man(2)

sh_new_variable_message

Controls a debugging feature for tracing the creation of new variables.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

The **sh_new_variable_message** variable controls a debugging feature for tracing the creation of new variables. Its primary debugging purpose is to catch the misspelling of an application-owned global variable. When set to **true**, an informational message (CMD-041) is displayed when a variable is defined for the first time at the command line. When set to **false**, no message is displayed.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. See the **set_app_var command man page for details.**

Other variables, in combination with **sh_new_variable_message**, enable tracing of new variables in scripts and Tcl procedures.

Warning: This feature has a significant negative impact on CPU performance when used with scripts and Tcl procedures. This feature should be used only when developing scripts or in interactive use. When you turn on the feature for scripts or Tcl procedures, the application issues a message (CMD-042) to warn you about the use of this feature.

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

SEE ALSO

get_app_var(2)
set_app_var(2)
sh_new_variable_message_in_proc(3)
sh_new_variable_message_in_script(3)

sh_new_variable_message_in_proc

Controls a debugging feature for tracing the creation of new variables in a Tcl procedure.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **sh_new_variable_message_in_proc** variable controls a debugging feature for tracing the creation of new variables in a Tcl procedure. Its primary debugging purpose is to catch the misspelling of an application-owned global variable.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. Please see the **set_app_var** command man page for details.

Note that the **sh_new_variable_message** variable must be set to **true** for this variable to have any effect. Both variables must be set to **true** for the feature to be enabled. Enabling the feature simply enables the **print_proc_new_vars** command. In order to trace the creation of variables in a procedure, this command must be inserted into the procedure, typically as the last statement. When all of these steps have been taken, an informational message (CMD-041) is generated for new variables defined within the procedure, up to the point that the **print_proc_new_vars** commands is executed.

Warning: This feature has a significant negative impact on CPU performance. This should be used only when developing scripts or in interactive use. When you turn on the feature, the application issues a message (CMD-042) to warn you about the use of this feature.

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

SEE ALSO

get_app_var(2)
print_proc_new_vars(2)
set_app_var(2)
sh_new_variable_message(3)
sh_new_variable_message_in_script(3)

sh_new_variable_message_in_script

Controls a debugging feature for tracing the creation of new variables within a sourced script.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

The **sh_new_variable_message_in_script** variable controls a debugging feature for tracing the creation of new variables within a sourced script. Its primary debugging purpose is to catch the misspelling of an application-owned global variable.

Note that this debugging feature is superseded by the new **set_app_var** command. This command allows setting only application-owned variables. See the **set_app_var** command man page for details.

Note that the **sh_new_variable_message** variable must be set to **true** for this variable to have any effect. Both variables must be set to **true** for the feature to be enabled. In that case, an informational message (CMD-041) is displayed when a variable is defined for the first time. When **sh_new_variable_message_in_script** is set to **false** (the default), no message is displayed at the time that the variable is created. When the **source** command completes, however, you see messages for any new variables that were created in the script. This is because the state of the variables is sampled before and after the **source** command. It is not because of inter-command sampling within the script. So, this is actually a more efficient method to see if new variables were created in the script.

For example, given the following script a.tcl:

```
echo "Entering script"
set a 23
echo a = $a
set b 24
echo b = $b
echo "Exiting script"
```

When **sh_new_variable_message_in_script** is **false** (the default), you see the following when you source the script:

```
prompt> source a.tcl
Entering script
a = 23
b = 24
Exiting script
Information: Defining new variable 'a'. (CMD-041)
Information: Defining new variable 'b'. (CMD-041)
prompt>
```

Alternatively, when **sh_new_variable_message_in_script** is **true**, at much greater cost, you see the following when you source the script:

```
prompt> set sh_new_variable_message_in_script true
Warning: Enabled new variable message tracing -
```

```
        Tcl scripting optimization disabled. (CMD-042)
true
prompt> source a.tcl
Entering script
Information: Defining new variable 'a'. (CMD-041)
a = 23
Information: Defining new variable 'b'. (CMD-041)
b = 24
Exiting script
prompt>
```

Warning: This feature has a significant negative impact on CPU performance. This should be used only when developing scripts or in interactive use. When you turn on the feature, the application issues a message (CMD-042) to warn you about the use of this feature.

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

SEE ALSO

get_app_var(2)
set_app_var(2)
sh_new_variable_message(3)
sh_new_variable_message_in_proc(3)

sh_obsolete_is_error

Raise a Tcl error when an obsolete command is executed.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When set this variable causes a Tcl error to be raised when an obsolete command is executed. Normally only a warning message is issued.

Obsolete commands have no effect.

SEE ALSO

get_app_var(2)
set_app_var(2)

sh_output_log_file

This variable is used to name the file to record logfile output. Set the variable to the empty string to disable output capture.

TYPE

string

DEFAULT

""

DESCRIPTION

This variable can be used to save almost all console output to a log file. The log file is useful for bug reproduction and reporting.

The first time the variable is set to a valid filename, all previously logged output is written to the specified file, overwriting any previous contents of the file. All subsequent logged output is appended to the file unless the variable is later reset.

If the variable is later changed to an empty string, all logged output is disabled.

If the variable is later set to an non-empty string that is an invalid filename, the new value is ignored and the old value is restored.

If the variable is later set to an non-empty string that is an valid filename, all subsequent logged output is written to this file. Any previous contents of the file are overwritten.

The log file may not capture the banner text from the beginning of the session, and does not capture the stack trace that is printed following a fatal error. You can use the UNIX tee command or redirect the output to capture this information.

sh_product_version

Indicates the version of the application currently running.

TYPE

string

DEFAULT

" "

DESCRIPTION

This variable is a read-only variable set by the application to indicate the version of the application currently running.

sh_script_stop_severity

Indicates the error message severity level that would cause a script to stop running before it completes.

TYPE

string

DEFAULT

application specific

DESCRIPTION

When a script is run with the **source** command, there are several ways to get it to stop running before it completes. One is to use the **sh_script_stop_severity** variable. This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a script to stop.
- When set to **W**, the generation of one or more warning or error messages causes a script to stop.
- When set to **none**, the generation messages does not cause the script to stop.

Note that **sh_script_stop_severity** is ignored if **sh_continue_on_error** is set to **true**.

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

SEE ALSO

get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)

sh_source_emits_line_numbers

Indicates the error message severity level that causes an informational message to be issued, listing the script name and line number where that message occurred.

TYPE

string

DEFAULT

application specific

DESCRIPTION

When a script is executed with the **source** command, error and warning messages can be emitted from any command within the script. Using the **sh_source_emits_line_numbers** variable, you can help isolate where errors and warnings are occurring.

This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a CMD-082 informational message to be issued when the command completes, giving the name of the script and the line number of the command.
- When set to **W**, the generation of one or more warning or error messages causes a the CMD-082 message.

The setting of **sh_script_stop_severity** affects the output of the CMD-082 message. If the setting of **sh_script_stop_severity** causes a CMD-081 message, then it takes precedence over CMD-082.

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

SEE ALSO

get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)
sh_script_stop_severity(3)
CMD-081(n)
CMD-082(n)

sh_source_logging

Indicates if individual commands from a sourced script should be logged to the command log file.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When you source a script, the **source** command is echoed to the command log file. By default, each command in the script is logged to the command log file as a comment. You can disable this logging by setting **sh_source_logging** to **false**.

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

SEE ALSO

get_app_var(2)
set_app_var(2)
source(2)

sh_source_uses_search_path

Causes the source command to use the search_path variable to search for files.

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to cause the **sourc** command to use the **search_path** variable when searching for files.

By default, the **source** command considers its file argument literally. By setting **sh_source_uses_search_path** to "true", the source command uses the **search_path** variable to search for files.

To change the value of this variable, enter **set sh_source_uses_search_path** *value*, where *value* is "false" or "true".

SEE ALSO

search_path(3)
source(2)

sh_tcllib_app_dirname

Indicates the name of a directory where application-specific Tcl files are found.

TYPE

string

DESCRIPTION

The **sh_tcllib_app_dirname** variable is set by the application to indicate the directory where application-specific Tcl files and packages are found. This is a read-only variable.

SEE ALSO

get_app_var(2)

sh_user_man_path

Indicates a directory root where you can store man pages for display with the **man** command.

TYPE

list

DEFAULT

empty list

DESCRIPTION

The **sh_user_man_path** variable is used to indicate a directory root where you can store man pages for display with the **man** command. The directory structure must start with a directory named *man*. Below *man* are directories named *cat1*, *cat2*, *cat3*, and so on. The **man** command will look in these directories for files named *file.1*, *file.2*, and *file.3*, respectively. These are pre-formatted files. It is up to you to format the files. The **man** command effectively just types the file.

These man pages could be for your Tcl procedures. The combination of defining help for your Tcl procedures with the **define_proc_attributes** command, and keeping a manual page for the same procedures allows you to fully document your application extensions.

The **man** command will look in **sh_user_man_path** after first looking in application-defined paths. The user-defined paths are consulted only if no matches are found in the application-defined paths.

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

SEE ALSO

define_proc_attributes(2)
get_app_var(2)
man(2)
set_app_var(2)

signature_analysis

Specifies whether signature analysis matching is enabled

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to disable all signature analysis.

By default, signature analysis is used during matching and verification. It can be runtime intensive and may not always be required.

When this variable is **true**, the individual signature_analysis_match* variables are used to control the types of signature analysis. See the list of variables in the **SEE ALSO** section. When this variable is **false**, all signature_analysis_match* variables are ignored and no signature analysis will occur.

SEE ALSO

signature_analysis_match_blackbox_input(3)
signature_analysis_match_blackbox_output(3)
signature_analysis_match_compare_points(3)
signature_analysis_match_datapath(3)
signature_analysis_match_hierarchy(3)
signature_analysis_match_net(3)
signature_analysis_match_pin_net(3)
signature_analysis_match_primary_input(3)
signature_analysis_match_primary_output(3)

signature_analysis_allow_net_match

Specifies whether signature analysis utilizes net-based matching methods

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to control whether signature analysis utilizes net-based matching methods.

By default, signature analysis does not utilize net-based matching methods. To force signature analysis to utilizenet-based matching methods, set this variable to "true".

To change the value of this variable, enter **set fBsignature_analysis_allow_net_match** *value*, where *value* is "true" or "false".

SEE ALSO

signature_analysis(3)
name_match_based_on_nets(3)
name_match_allow_subset_match(3)
signature_analysis_allow_subset_match(3)
signature_analysis_match_primary_input(3)
signature_analysis_match_primary_output(3)

signature_analysis_allow_subset_match

Specifies whether signature analysis utilizes subset matching methods

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to control whether signature analysis utilizes subset matching methods.

By default, signature analysis utilizes subset matching methods. To force signature analysis to not utilize subset matching methods, set this variable to "false".

To change the value of this variable, enter **set fBsignature_analysis_allow_subset_match** *value*, where *value* is "true" or "false".

SEE ALSO

signature_analysis(3)
name_match_allow_subset_match(3)
name_match_based_on_nets(3)
signature_analysis_allow_net_match(3)
signature_analysis_match_primary_input(3)
signature_analysis_match_primary_output(3)

signature_analysis_match_blackbox_input

Specifies whether signature analysis attempts to match previously unmatched black box inputs

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to control whether signature analysis attempts to match previously unmatched black box inputs.

By default, signature analysis attempts to match previously unmatched black box inputs. To force signature analysis to not attempt to match black box inputs, set this variable to "false".

To change the value of this variable, enter **set signature_analysis_match_blackbox_input** *value*, where *value* is "true" or "false".

SEE ALSO

signature_analysis(3)
signature_analysis_match_blackbox_output(3)
signature_analysis_match_primary_input(3)
signature_analysis_match_primary_output(3)

signature_analysis_match_blackbox_output

Specifies whether signature analysis attempts to match previously unmatched black box outputs

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to control whether signature analysis attempts to match previously unmatched black box outputs.

By default, signature analysis attempts to match previously unmatched black box outputs. To force signature analysis to not attempt to match black box outputs, set this variable to "false".

To change the value of this variable, enter **set signature_analysis_match_blackbox_output** *value*, where *value* is "true" or "false".

SEE ALSO

signature_analysis(3)
signature_analysis_match_blackbox_input(3)
signature_analysis_match_primary_input(3)
signature_analysis_match_primary_output(3)

signature_analysis_match_compare_points

Specifies whether signature analysis attempts to match compare points.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether signature analysis attempts to match the previously unmatched compare points.

By default, signature analysis attempts to match previously unmatched compare points. To prevent signature analysis from attempting to match the previously unmatched compare points, set this variable to **false**.

SEE ALSO

signature_analysis(3)
signature_analysis_match_datapath(3)
signature_analysis_match_hierarchy(3)

signature_analysis_match_datapath

Specifies whether signature analysis attempts to match datapath blocks and their pins.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether signature analysis attempts to match previously unmatched datapath blocks and their pins.

By default, signature analysis attempts to match previously unmatched datapath blocks and their pins. To prevent signature analysis to match previously unmatched datapath blocks and their pins, set this variable to **false**.

If you set the **signature_analysis_match_hierarchy** variable to **true**, signature analysis is used to match previously unmatched datapath blocks and their pins independent of the value of the **signature_analysis_match_datapath** variable .

SEE ALSO

signature_analysis(3)
signature_analysis_match_compare_points(3)
signature_analysis_match_hierarchy(3)

signature_analysis_match_hierarchy

Specifies whether signature analysis attempts to match hierarchical blocks and their pins.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether signature analysis attempts to match the previously unmatched hierarchical blocks and their pins.

By default, signature analysis attempts to match previously unmatched hierarchical blocks and their pins. To prevent signature analysis from attempting to match the previously unmatched hierarchical blocks and their pins, set this variable to **false**.

When the **signature_analysis_match_hierarchy** variable is set to **false**, signature analysis might still be used to match previously unmatched datapath blocks and their pins if the **signature_analysis_match_datapath** variable is set to **true**.

When the **signature_analysis_match_hierarchy** variable is set to **true**, signature analysis matches previously unmatched datapath blocks and their pins irrespective of the value of the **signature_analysis_match_datapath** variable.

SEE ALSO

signature_analsis(3)
signature_analysis_match_compare_points(3)
signature_analysis_match_datapath(3)

signature_analysis_match_net

Specifies whether signature analysis attempts to match reference nets to implementation nets.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls whether signature analysis attempts to match previously unmatched reference nets to implementation nets. Such matches identify potential cutpoints and may improve the verification performance.

By default, signature analysis attempts to match previously unmatched reference nets to implementation nets. To prevent signature analysis from attempting to match previously unmatched reference nets to implementation nets, set this variable to **false**.

SEE ALSO

signature_analysis(3)
signature_analysis_match_compare_points(3)
signature_analysis_match_datapath(3)
signature_analysis_match_hierarchy(3)
signature_analysis_match_pin_net(3)
name_match_net(3)

signature_analysis_match_pin_net

Specifies whether signature analysis attempts to match hierarchical pins to nets.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls whether signature analysis attempts to match previously unmatched hierarchical pins in one design to nets in the other design. Such matches identify potential cutpoints and might improve verification performance.

By default, signature analysis attempts to match previously unmatched hierarchical pins to nets. To prevent signature analysis from attempting to match previously unmatched hierarchical pins to nets, set this variable to **false**.

SEE ALSO

signature_analysis(3)
signature_analysis_match_compare_points(3)
signature_analysis_match_datapath(3)
signature_analysis_match_hierarchy(3)
signature_analysis_match_net(3)
name_match_pin_net(3)

signature_analysis_match_primary_input

Specifies whether signature analysis attempts to match previously unmatched primary inputs

TYPE

string

DEFAULT

"true"

DESCRIPTION

Use this variable to control whether signature analysis attempts to match previously unmatched primary inputs.

By default, signature analysis attempts to match previously unmatched primary inputs. To force signature analysis to not attempt to match primary inputs, set this variable to "false".

To change the value of this variable, enter **set signature_analysis_match_primary_input** *value*, where *value* is "true" or "false".

SEE ALSO

signature_analsis(3)
signature_analysis_match_primary_output(3)

signature_analysis_match_primary_output

Specifies whether signature analysis attempts to match previously unmatched primary outputs

TYPE

string

DEFAULT

"false"

DESCRIPTION

Use this variable to control whether signature analysis attempts to match previously unmatched primary outputs.

By default, signature analysis attempts to match previously unmatched primary outputs. To force signature analysis to not attempt to match primary outputs, set this variable to "false".

To change the value of this variable, enter **set signature_analysis_match_primary_output** *value*, where *value* is "true" or "false".

SEE ALSO

signature_analysis(3)
signature_analysis_match_primary_input(3)

svf_allow_rtl_file_map

Configuration for the SVF Reuse flow. Specifies pairings to match up RTL filenames as read by Formality ("eco" files) vs. filenames as specified in guide_file_info entries in SVF ("original" files). These pairings override the ones automatically found in the flow.

TYPE

list of lists of strings

SYNTAX

set svf_allow_rtl_file_map { { <ecofile1> <origfile1> } { <ecofile2> <origfile2> } ... }

DEFAULT

None

DESCRIPTION

The SVF Reuse flow allows older SVF to be applied against slightly altered versions of RTL files. This is done by means of the SVF containing information about each file as of the time it was generated (given via guide_file_info SVF commands). That is compared against the corresponding files read in by Formality, and corrections are applied.

A key part of this flow is correct matching of the files read by Formality (which we refer to as the "eco" files, as they may contain modest alterations), vs. the files whose info is given in the SVF (the "original" files). This is done automatically by employing several methods, and normally should be correct.

But to guard against the possibility that the automatic matchups are done incorrectly, we provide a manual override mechanism via this variable. The user specifies the overrides as a list of eco-to-original file pairings (see example below). The names given are the base filenames and any number of directory levels above.

Note that an additional matchup issue presents itself for this override mechanism itself. We do not require that the entire given path to an eco filename match to the entire path of the file that Formality read in (the same applies to the given "orig" file paths vs. the filepath info from the SVF). Instead, we only require that the base file name matches, and then we use as many of the directory levels up from there as we need to disambiguate. We print error messages if we cannot disambiguate, or if we fail to find a match for the eco or orig file patsh.

EXAMPLE

Example situation:

- Formality has read in files: dir1/dir2/foo.altered.v dir1/dir100/foo.altered.v

- SVF data has info for these files: dirA/dirB/foo.v dirX/dirY/foo.v

- We may specify a matchup overide as thus: set svf_allow_rtl_file_map { \ { dir2/foo.altered.v dirB/foo.v } \ { dir100/foo.altered.v dirY/foo.v } \ }

This will cause the eco-to-orig matchups to be forced as follows: dir1/dir2/foo.altered.v --> dirA/dirB/foo.v dir1/dir100/foo.altered.v --> dirX/dirY/foo.v

SEE ALSO

guide_file_info(2)

svf_breakpoint

Enables processing of SVF guide_breakpoint commands and supports the breakpoint verification flow.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

Use this variable to enable or disable processing of guide_breakpoint commands and enable the breakpoint verification flow. Values are "true", "false", 0, 1.

SEE ALSO

load_breakpoint_data(2)
guide_breakpoint(2)

svf_checkpoint

Enables processing of SVF guide_checkpoint commands.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to enable or disable processing of any guide_checkpoint commands. values are "true", "false", 0, 1.

SEE ALSO

svf_checkpoint_save_session(3)
svf_checkpoint_stop_when_rejected(3)

svf_checkpoint_auto_setup_commands

Controls which user setup types will be automatically shared with checkpoint verifications.

TYPE

TCL list: Legal values: "set_black_box", "set_constant", "set_constraint", "set_cutpoint", "set_dont_verify_points", "none", "all"

DEFAULT

""

DESCRIPTION

Use this variable to control which kinds of user setup Formality will share during checkpoint verification. By default, no user setup is shared with checkpoints (or any other SVF guides) during preverify unless it is specified to be shared. Note, users should not change any setup used during preverification once preverify has completed. Shortcuts "all" and "none" may also be used. Note, "none" cannot be specified with any other setup choices.

EXAMPLE

fm_shell (setup)> set_app_var svf_checkpoint_auto_setup_commands set_constant
set_constant

Any constant setup users specify before preverify will be shared with the checkpoint verification during preverify.

fm_shell (setup)> echo $svf_checkpoint_auto_setup_commands
set_constant

fm_shell (setup)> lappend svf_checkpoint_auto_setup_commands set_dont_verify_points
set_constant set_dont_verify_points

Adds set_dont_verify_points to the list of setup users can specify before preverify that will be shared with the checkpoint verification during preverify.

fm_shell (setup)> lappend svf_checkpoint_auto_setup_commands all
all

All legal setup types will be shared during checkpoint perverify (shorter than enumerating them all).

fm_shell (setup)> set_app_var svf_checkpoint_auto_setup_commands ""

Don't share any setup with checkpoint verifications.

fm_shell (setup)> set_app_var svf_checkpoint_auto_setup_commands "none"

Another way to prevent sharing any setup with checkpoint verifications.

svf_checkpoint_format_verilog

Controls the checkpoint types that are verified against their Verilog implementation.

TYPE

Tcl list of checkpoint types, or true for all eligible checkpoints.

DEFAULT

""

DESCRIPTION

Use this variable to control the checkpoint types that are verified against their Verilog implementation. By default, checkpoints are verified against their NDM implementation. Checkpoints types can be specified with the -type field of the checkpoint guide (for example, ckpt_logic_opt), or set to true if Verilog is to be used for all eligible checkpoints.

Note: There is no Verilog netlist for the pre_map checkpoint.

EXAMPLE

```
fm_shell (setup)> set svf_checkpoint_format_verilog ckpt_logic_opt
ckpt_logic_opt
```

This example specifies Verilog format for all logic_opt checkpoints in the SVF.

```
fm_shell (setup)> set svf_checkpoint_format_verilog true
true
```

This example specifies Verilog format for all checkpoints in the SVF, except for ckpt_pre_map, if it exists.

svf_checkpoint_run_analyze_points

TYPE

string

DEFAULT

"ckpt_pre_retime"

DESCRIPTION

This variable controls if a given rejected guide_checkpoints runs analyze_points before terminating. Allowed values are ckpt_pre_retime, false and all.

ckpt_pre_retime: Runs analyze_points after the ckpt_pre_retime checkpoint rejection.

false: Does not run analyze_points after rejected checkpoints.

all: Runs analyze_points after every checkpoint rejection.

svf_checkpoint_save_session

Controls when session files capturing checkpoint verifications are generated.

TYPE

string: Values "not_passed", "failed", "inconclusive", "all", "none"

DEFAULT

"not_passed"

DESCRIPTION

Generates a session file of the checkpoint verifcation. The session file is saved in ./fm_checkpoint_sessions[<n>]/checkpoint_<checkpoint_id>.fss . To change the control of when session files are saved , enter **set svf_checkpoint_save_session** *value*, where *value* is one of the following: "not_passed", "failed", "inconclusive", "all" or "none".

* "not_passed" - save session file if result of checkpoint verification is anything but successful

* "failed" - save session file only if result of checkpoint verification is failed

* "inconclusive" - save session file only if result of checkpoint verification is inconclusive

* "all" - save session file regardless of checkpoint verification result

* "none" - do not save checkpoint verification session file

SEE ALSO

svf_checkpoint(3)
svf_checkpoint_stop_when_rejected(3)

svf_checkpoint_stop_when_rejected

Enables early termination of svf processing as soon as a guide_checkpoint command is rejected.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

When enabled, svf processing will stop and return to setup mode when a guide_checkpoint is rejected. The reference design will have partially applied SVF. Retyping match or verify will cause the run to continue processing svf from where it left off. values are "true", "false", 0, 1.

SEE ALSO

svf_checkpoint(3)
svf_checkpoint_save_session(3)

svf_datapath

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether Formality will process all **guide_datapath, guide_divider_netlist, guide_merge, guide_multiplier, guide_replace, guide_transformation** commands found in the user specified SVF file. A value of "true" indicates to Formality that these commands should be processed. A value of "false" indicates that these commands will not be processed.

Note that the Presto reader in Design Compiler must be used in order for this Enhanced Datapath flow to work properly. For VHDL, the following two variables must be set to "true" in your Formality run to ensure compatibility with Presto:

```
set hdlin_vhdl_presto_naming true
set hdlin_vhdl_presto_shift_div true
```

SEE ALSO

guide_datapath(2)
guide_divider_netlist(2)
guide_merge(2)
guide_multiplier(2)
guide_replace(2)
guide_transformation(2)
hdlin_vhdl_presto_naming(3)
hdlin_vhdl_presto_shift_div(3)
set_svf(2)

svf_guide_constant_acceleration

Enables an alternate strategy for verifying guide_constant commands during the preverify stage

TYPE

String

DEFAULT

source

DESCRIPTION

This variable enables an alternate strategy for verifying **guide_constant** commands during the preverify stage. This strategy is significantly faster than the default strategy. Allowed values for this variable are as follows:

- source (default) - Enables the alternate strategy for source guide_constant commands only
- false - Disables the alternate strategy

SEE ALSO

guide_constant(2)

svf_guide_constraints_acceleration

Enables an alternate strategy for verifying guide_constraints commands during the preverify stage

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable enables an alternate strategy for verifying **guide_constraints** commands during the preverify stage. This strategy is significantly faster than the default strategy. Allowed values for this variable are as follows:

- true (default) - Enables the alternate strategy for guide_constraints commands
- false - Disables the alternate strategy

SEE ALSO

guide_constraints(2)

svf_guide_reg_acceleration

Enables an alternate strategy for verifying register guides during the preverify stage

TYPE

String

DEFAULT

true

DESCRIPTION

This variable enables an alternate strategy for verifying the **guide_reg_constant** and **guide_reg_merging** commands during the preverify stage. This strategy is significantly faster than the previous default strategy. Allowed values for this variable are as follows:

- constant - Enables the alternate strategy for the guide_reg_constant command only.
- true (the default) - Enables the alternate strategy for both the guide_reg_constant and guide_reg_merging commands.
- false - Disables the alternate strategy.

SEE ALSO

guide_reg_constant(2)
guide_reg_merging(2)

svf_ignore_unqualified_fsm_information

Specifies to ignore or not ignore the **guide_fsm_reencoding** command in SVF files.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to ignore or not ignore the **guide_fsm_reencoding** command in SVF files.

To change the value of this variable, enter **set svf_ignore_unqualified_fsm_information** "*value*", where *value* is "true" or "false".

Note: If you change the value of this variable, only SVF files you read in thereafter are affected by the change.

The **guide_fsm_reencoding** command includes unqualified information regarding the state optimization if some possible state encodings are not used. They will be treated as don't cares during optimization. State machines which use all possible state values will not be affected by this variable.

svf_inv_push

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether Formality will process all **guide_inv_push** commands found in the user specified SVF file. A value of "true" indicates to Formality that **guide_inv_push** commands will be processed. A value of "false" indicates that**guide_inv_push** commands will be ignored. Only **guide_inv_push** commands are affected by this variable.

SEE ALSO

set_svf(2)

svf_port_constant

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether Formality will process all **guide_port_constant** commands found in the user specified automated setup file (SVF). A value of "true" indicates to Formality that **guide_port_constant** commands should be accepted. A value of "false" indicates that **guide_port_constant** commands will not be processed. No guide commands other than **guide_port_constant** commands are affected. Note that **synopsys_auto_setup** must also be "true" for **guide_port_constant** to be accepted.

SEE ALSO

guide_port_constant(2)
set_svf(2)
synopsys_auto_setup(3)

svf_report_guidance_write_design_data

Controls if svf design data is written to formality_svf and output of report_guidance -to.

TYPE

string: Values "true", "false"

DEFAULT

"true"

DESCRIPTION

When set to false, disables writing of checkpoints, and netlists to save disk space. The resulting file can not be use as input to future runs.

SEE ALSO

report_guidance(3)

svf_retiming

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether Formality will process all **guide_retiming** commands found in the user specified SVF file. A value of "true" indicates to Formality that **guide_retiming** commands will be processed. A value of "false" indicates that**guide_retiming** commands will be ignored. Only **guide_retiming** commands are affected by this variable.

If the **set_parameters -retimed** command has been set on a design, then all**guide_retiming** commands will be ignored regardless of how **svf_retiming** is set.

SEE ALSO

set_svf(2)
set_parameters(2)

svf_scan

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether Formality will process all **guide_scan_input** commands found in the user specified SVF file. A value of "true" indicates to Formality that **guide_scan_input** commands should be accepted. A value of "false" indicates that **guide_scan_input** commands will not be processed. No guide commands other than **guide_scan_input** commands are affected. Note that **synopsys_auto_setup** must also be "true" for **guide_scan_input** to be accepted.

SEE ALSO

guide_scan_input(2)
set_svf(2)
synopsys_auto_setup(3)

svr_keep_supply_nets

Enables the Verilog netlist reader to separate supply nets from logic nets.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to enable Formality's Verilog netlist reader to separate supply nets from logic nets. For example VDD and VSS could be separately handled by using this variable:

```
module test ( A, Y );
   input A;
   output Y;

   supply1 vdd;
   supply0 vss;
   PVDD3P c0 ( .TAVDD(vdd) );
   AND2X2 c1 ( .A(A), .B(1'b1), .Y(Y) );
endmodule
```

symbol_library

Specifies the symbol .sdb libraries to use during schematic generation.

TYPE

string

DEFAULT

DESCRIPTION

Use this variable to specify the .sdb library files to search for symbols during schematic generation. This variable is a list of full path names to .sdb library files.

Symbols are located by searching the library files specified by **symbol_library** first, in the order they are listed, next in the library files residing in the directories of the associated design files.

synopsys_auto_setup

Enables the Synopsys Auto Setup Mode.

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

This variable enables the Synopsys Auto Setup Mode in Formality.

By default, the Synopsys auto setup mode sets the appropriate Formality commands and variables to account for external constraints and RTL interpretation considered by Synopsys implementation tools , such as Design Compiler. To use this mode, set the **synopsys_auto_setup** variable to **true** before running the **set_svf** command.

When the **synopsys_auto_setup_mode** variable is set to **true**, the following Formality variables and commands are changed as shown:

- **hdlin_ignore_embedded_configuration** = true
- **hdlin_ignore_full_case** = false
- **hdlin_ignore_parallel_case** = false
- **signature_analysis_allow_subset_match** = false
- **svf_ignore_unqualified_fsm_information** = false (only changed when there are **guide_fsm_reencoding** commands in the sutomated setup file for verification (SVF))
- **upf_assume_related_supply_default_primary** = true
- **upf_use_additional_db_attributes** = true
- **verification_set_undriven_signals** = synthesis
- **verification_verify_directly_undriven_output** = false
- **set_mismatch_message_filter -warn**
- **hdlin_enable_verilog_configurations_array_n_block** = true
- **svf_checkpoint_auto_setup_commands** = all

You can override the above settings individually after the **synopsys_auto_setup_mode** variable is set, or use the variable **synopsys_auto_setup_filter** before setting **synopsys_auto_setup**.

If the **synopsys_auto_setup** variable is set to **true** and the **set_svf** command is used to read in the automated setup file (SVF), the additional setup information (external constraints) is passed to Formality through the SVF file. The constraint actions include:

Enable **guide_environment {{ clock_gating ... }}**
  (set verification_clock_gate_hold_mode ...)
Enable **guide_environment {{ hdlin_dyn_array_bnd_check ... }}**
  (set hdlin_dyn_array_bnd_check ...)
Enable **guide_environment {{ hdlin_optimize_enum_types ... }}**
  (set hdlin_optimize_enum_types ...)
Enable **guide_environment {{ hdlin_infer_enumerated_types ... }}**
  (set hdlin_infer_enumerated_types ...)
Enable **guide_port_constant**
  (set_constant, for DC set_logic_one/zero/dc)
Enable **guide_scan_input**
  (set_constant on scan enable to disable scan)
Enable **guide_set_rounding**
  (set external and internal rounding positions on multipliers)

The summary is reported and included in the Formality transcript listing all the non-default variable settings and external constraint information passed using the automated setup file (SVF).

EXAMPLES

The following example shows how to enable the Synopsys auto setup mode.

```
fm_shell (setup)> set synopsys_auto_setup true
true
fm_shell (setup)> set_svf default.svf
SVF set to '/remote/misc11/user1/default.svf'.
1
```

The following example shows how to enable the Synopsys auto setup mode, and change the default for handling the undriven signals.

```
fm_shell (setup)> set synopsys_auto_setup true
true
fm_shell (setup)> set_svf default.svf
SVF set to '/remote/misc11/user1/default.svf'.
1
fm_shell (setup)> printvar verification_set_undriven_signals
verification_set_undriven_signals = "synthesis"
fm_shell (setup)> set verification_set_undriven_signals BINARY:X
BINARY:X
```

SEE ALSO

synopsys_auto_setup_filter(3)
hdlin_dyn_array_bnd_check(3)
hdlin_ignore_embedded_configuration(3)
hdlin_ignore_full_case(3)
hdlin_ignore_parallel_case(3)
hdlin_infer_enumerated_types(3)
hdlin_optimize_enum_types(3)
set_constant(2)
set_mismatch_message_filter(2)
remove_mismatch_message_filter(2)
signature_analysis_allow_subset_match(3)
svf_ignore_unqualified_fsm_information(3)
svf_scan(3)
upf_assume_related_supply_default_primary(3)
upf_use_additional_db_attributes(3)
verification_clock_gate_hold_mode(3)
verification_set_undriven_signals(3)
verification_verify_directly_undriven_output(3)
svf_checkpoint_auto_setup_commands(3)

synopsys_auto_setup_filter

Enables selective control of the variables that are set by the **synopsys_auto_setup** variable.

TYPE

list

DEFAULT

""

DESCRIPTION

This variable enables selective control of the variables that are set by the **synopsys_auto_setup** variable. Specify one or more of the following values:

- **clock_gating** - ignores the **clock_gating latch_and**, **clock_gating latch_or**, **clock_gating and**, and **clock_gating or** commands
- **hdlin_dyn_array_bnd_check**
- **hdlin_ignore_embedded_configuration**
- **hdlin_ignore_full_case**
- **hdlin_ignore_parallel_case**
- **mismatch_message**
- **rounding** - ignores all **guide_set_rounding** commands
- **scan_input** - ignores all **guide_scan_input** and **guide_dont_verify_scan_input** commands
- **signature_analysis_allow_subset_match**
- **svf_ignore_unqualified_fsm_information**
- **upf_assume_related_supply_default_primary**
- **upf_use_additional_db_attributes**
- **verification_set_undriven_signals**
- **verification_verify_directly_undriven_output**

When these variables listed are specified using the **synopsys_auto_setup_filter** variable, they are not changed from their default when the **synopsys_auto_setup** variable is set to **true**.

Set the **synopsys_auto_setup_filter** variable before setting the **synopsys_auto_setup** variable to **true** or the **set_svf** command.

You can still override any individual variable setting of the **synopsys_auto_setup** mode by setting any of these variables after the **synopsys_auto_setup** variable has been set.

EXAMPLES

The following example allows all default options of auto setup mode except for the scan insertion setup:

```
fm_shell (setup)> set synopsys_auto_setup_filter {scan_input}
scan_input
fm_shell (setup)> set synopsys_auto_setup true
true
fm_shell (setup)>  set_svf default.svf
SVF set to '/remote/misc11/user1/default.svf'.
1
```

The following example allows all default options of the auto setup mode except for synthesis interpretation of full_case and parallel_case directives:

```
fm_shell (setup)> set synopsys_auto_setup_filter { hdlin_ignore_full_case
    hdlin_ignore_parallel_case }
fm_shell (setup)> set synopsys_auto_setup true
true
fm_shell (setup)>  set_svf default.svf
SVF set to '/remote/misc11/user1/default.svf'.
1
fm_shell (setup)> printvar hdlin_ignore_full_case
hdlin_ignore_full_case = "true"
fm_shell (setup)> printvar hdlin_ignore_parallel_case
hdlin_ignore_parallel_case = "true"
```

SEE ALSO

synopsys_auto_setup(3)
clock_gate_hold_mode(3)
guide_set_rounding(2)
hdlin_dyn_array_bnd_check(3)
set_mismatch_message_filter(2)
hdlin_ignore_embedded_configuration(3)
hdlin_ignore_full_case(3)
hdlin_ignore_parallel_case(3)
hdlin_infer_enumerated_types(3)
hdlin_optimize_enum_types(3)
set_constant(2)
signature_analysis_allow_subset_match(3)
svf_ignore_unqualified_fsm_information(3)
upf_assume_related_supply_default_primary(3)
upf_use_additional_db_attributes(3)
verification_set_undriven_signals(3)
verification_verify_directly_undriven_output(3)

synopsys_program_name

Indicates the name of the program currently running.

TYPE

string

DESCRIPTION

This variable is read only, and is set by the application to indicate the name of the program you are running. This is useful when writing scripts that are mostly common between some applications, but contain some differences based on the application.

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

SEE ALSO

get_app_var(2)

synopsys_root

Specifies the value of the $SYNOPSYS environment variable.

TYPE

string

DEFAULT

" "

DESCRIPTION

This variable is a read-only variable set by the application to indicate the value of the $SYNOPSYS environment variable.

template_naming_style

Generates automatically a unique name when a module is built.

TYPE

string

DEFAULT

%s_%p

DESCRIPTION

Generates automatically a unique name when a module is built. This variable is one of three string variables that determine the naming conventions for parameterized modules (templates) built into a design through the set_top command. The unique name automatically generated uses the module name, parameter names, and parameter values.

The template_naming_style variable determines what character or characters appear with the design name and the parameter name. The string value must contain %s, which stands for the name of the original design, and %p, which stands for the name and value of the parameter or parameters. You can optionally include any ASCII character or characters, or none, with the %s and %p. For example, for a design named **DesignName** that has a parameter **parm1**, the default %s_%p causes the name **DesignName_parm1** to be generated.

Further, %s$%p, %s_*_%p, and %s%p would generate respectively the names **DesignName$parm1**, **DesignName_*_parm1**, and **DesignNameparm1**.

If a design has a noninteger parameter (or if template_naming_style = ""), the following definitions are locked down for these variables:

template_naming_style = %s_%p template_parameter_style = %d template_separator_style = _

SEE ALSO

hdl_naming_threshold(3)
template_parameter_style(3)
template_separator_style(3)

template_parameter_style

Generates automatically a unique name when a module is built.

SH TYPE string

DEFAULT

%s%d

DESCRIPTION

Generates automatically a unique name when a module is built. This variable is one of three string variables that determine the naming conventions for parameterized modules (templates) built into a design through the set_top command. The unique name automatically generated uses the module name, parameter names, and parameter values.

The template_parameter_style variable determines what character or characters appear with the parameter name and its value.

The string may contain %s, which stands for the name of the parameter, and %d, which stands for the value of the parameter. You can optionally include any ASCII character or characters, or none, with the %s and %d. For example, for a parameter named **parm** that has a value of **1**, the default %s%d causes the name **parm1** to be generated.

Other examples:

template_naming_style = "%s$%p" template_parameter_style = %s_%d results in **DesignName$parm_1**

template_naming_style = "%s_%p" template_parameter_style = %s@%d results in **DesignName_parm@1**

If a design has a noninteger parameter (or if template_naming_style = ""), the following definitions are locked down for these variables:

template_naming_style = %s_%p template_parameter_style = %d template_separator_style = _

SEE ALSO

hdl_naming_threshold(3)
template_naming_style(3)
template_separator_style(3)

template_separator_style

Generates automatically a unique name when a module is built.

TYPE

string

DEFAULT

_

DESCRIPTION

Generates automatically a unique name when a module is built. This variable is one of three string variables that determine the naming conventions for parameterized modules (templates) built into a design through the set_top command. The unique name automatically generated uses the module name, parameter names, and parameter values.

The template_separator_style variable determines what character or characters appear with parameter names for templates that have more than one parameter. You can designate any ASCII character or characters, or none. The default value is an underscore (_).

For example, for a design called **DesignName** that has parameters named **parm1**, **parm2**, and **parm3**, if template_naming_style = "%s_%p" (the default value), and template_separator_style = "_", the name **DesignName_parm1_parm2_parm3** is generated.

Other examples:

template_naming_style = "%s$%p" template_separator_style = "_" results in **DesignName$parm1_parm2_parm3**

template_naming_style = "%s#%p" template_separator_style = "/" results in **DesignName#parm1**/**parm2**/**parm3**

If a design has a noninteger parameter (or if template_naming_style = ""), the following definitions are locked down for these variables:

template_naming_style = %s_%p template_parameter_style = %d template_separator_style = _

SEE ALSO

hdl_naming_threshold(3)
template_naming_style(3)
template_parameter_style(3)

upf_allow_bias_pin_connection

Specifies the type of bias pg_pin (supply pin) to use with the **connect_supply_net** command.

TYPE

string

DEFAULT

"all"

DESCRIPTION

This variable specifies the type of bias pg_pin (supply pin) to use with the **connect_supply_net** command.

When the UPF **connect_supply_net** command attempts a connection to any bias pg_pin on a technology library cell, the behavior of the tool depend on the value of this variable. Specify one of the following values for the **upf_allow_bias_pin_connection** variable:

- **routing_pin** - allow connection to bias pg_pins with attribute physical_connection : routing_pin
- **all** - allow connection to bias pg_pins with attribute physical_connection : routing_pin or physical_connection : device_layer
- **none** - report an error if the**connect_supply_net** command connects to any type of bias pg_pin

If the direction of the bias pg_pin is output or internal, the tool reports an error.

SEE ALSO

load_upf(2)

upf_allow_connect_logic_net_cross_pd

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

Prior to UPF 3.1, it was an error when processing connect_logic_net if the connection from the specified net in the active UPF scope to any element in port_list crosses any power-domain boundaries. When this variable is set to *true*, this check is disabled. Otherwise, the tool issues an error when it detects a domain crossing.

The variable must be set before issuing the **load_upf** command.

SEE ALSO

load_upf(2)

upf_allow_domain_merging

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

When this variable is set to *true*, the tool attempt sto automatically merge duplicate power domains specified for the same design element. Otherwise, the tool issues errors when it detects duplicate specifications.

Note that the domains are merged only if they are behaviorally equivalent. This includes primary supply nets, isolation strategies, retention strategies and power switches.

This feature is intended to support hierarchical design flows where the power behavior for individual blocks is specified in both the block and the full chip UPF files.

The variable must be set before issuing the **load_upf** command.

SEE ALSO

load_upf(2)

upf_allow_mixed_retention_cell_type

This variable allows the tool to map to zero-pin retention cells and implement necessary clamp cells for any retention strategies which specify both zero-pin and other retention registers. This variable is ignored when the **upf_use_library_cells_for_retention** variable is set to false (default).

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When the **upf_use_library_cells_for_retention** variable is set to true along with the default of the **upf_allow_mixed_retention_cell_type** variable (true), the Formality tool ignores the value of the **upf_infer_clamp_for_retention** variable and maps to zero-pin retention and infers appropriate clamp cells (based on the attribute information found in the zero-pin library cell definition) during the **load_upf** stage if there is an appropriate zero-pin library cell listed with the **map_retention_cell -lib_cells {}** command. If an appropriate zero-pin retention cell cannot be found, the tool uses cells with save or restore pins. Retention strategies that have unique save and restore signals cannot be mapped to a zero-pin retention cell and are always mapped to two-pin retention cells.

SEE ALSO

upf_use_library_cells_for_retention(3)
upf_infer_clamp_for_retention(3)
load_upf(2)

upf_allow_rtl_pgnet_name_space_conflict

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

When this variable is set to *true*, Formality attempts to automatically merge supply net and port definitions in the UPF with pre-existing nets and ports of the same names in the target design. Otherwise, it issues errors when it detects duplicate specifications.

The variable must be set before issuing the **load_upf** command.

SEE ALSO

load_upf(2)

upf_allow_unverified_write_power_model

TYPE

boolean

DEFAULT

true

DESCRIPTION

This variable controls whether Formality Power Models can be written without a successful verification.

When **upf_allow_unverified_write_power_model** is set to *true* (the default) power models can be written at any time after **set_top**. When it is set to *false* power models can only be written after a successful verification thus preventing inadvertent use of an invalid model elsewhere in the flow.

SEE ALSO

write_power_model(2)

upf_assume_lower_domain_boundary

Controls whether Formality automatically uses lower domain boundaries during upf processing.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls how Formality determines domain boundary ports when there are nested power domains. When this variable is true , Formality will assume the ports of a nested power domain are domain boundary ports for the parent power domain. This is compatible with the IEEE 1801-2015 Standard. When false (the default), Formality will not treat nested domain ports as boundary ports for the parent.

For backward compatiblity with older versions of tools which did not use the UPF lower domain boundary by default, set this variable to false.

How this works is that this variable is the default assumption for the UPF design_attribute "lower_domain_boundary" when the top design does not have it explicitly set. So in the following example regardless of this variable, domain PDT will not have a lower boundary, but PDM will:

```
set_design_attributes -elements {.} -attribute lower_domain_boundary false
set_design_attributes -elements {MID} -attribute lower_domain_boundary true
create_power_domain PDT -include_scope
create_power_domain PDM -scope MID
create_power_domain PDB -scope MID/BOT
```

Instead of using design attribute "lower_domain_boundary", users are now encourged to use the strategy option of "-applies_to_boundary <lower|upper|both>" to control which part of a domain boundary a given strategy will be applied to.

SEE ALSO

load_upf(2)

upf_assume_related_supply_default_primary

Controls whether Formality should assume top-level & blackbox ports are related to primary supplies by default

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls how Formality handles related supplies for isolation insertion purposes on top-level & blackbox pins that do not have explicit related supplies defined. When true (the default), during **load_upf** isolation insertion, Formality will assume that a port which has no explicit driver/receiver supply set (or related_power/related_ground) is related to the domain primary supplies for the port(pin).

When this variable is false Formality will issue an error when analyzing isolation source/sink supplies at a top level port or blackbox pin with no related supplies.

This variable does not affect related supply corruption logic insertion for the top level ports/blackbox pins, it only affects isolation insertion.

In a Synopsys flow, when the **synopsys_auto_setup** variable is set to *true*, the **upf_assume_related_supply_default_primary** variable will also be set to *true*.

SEE ALSO

load_upf(2)
synopsys_auto_setup(3)
upf_disable_spa_corruption(3)

upf_auto_analyze

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether Formality automatically runs **analyze_upf** on both the reference and implementation containers before any verification.

If **analyze_upf** finds problems, it will issue an error and prevent verification from proceeding. If you wish to run verification when errors are reported (not recommended) or you run **analyze_upf** separately you can set this variable to false.

SEE ALSO

analyze_upf(2)

upf_auto_invert_ground_match

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether Formality automatically inverts name-based matches between ports/pins that are UPF ground ports/pins whose ON state is 1 and ports/pins that are not UPF ground ports/pins whose ON state is 1. By default, Formality inverts such matches.

SEE ALSO

upf_ground_logic_value(3)

upf_bbox_use_switch_ack

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When upf commands are loaded onto a black box, Formality will push the black box down a level and create a wrapper design in which to place the power intent. All ports of the original black box are wired to the new internal black box. When processing "create_power_switch" there is a question of how to handle declared "ack" port connections. This variable controls that processing. When set to 'true', Formality will disconnect the specified output port from the internal black box and connect it to the specified switch ack pin. When the variable is set to 'false', Formality will leave the specified output port connected to the internal black box and leave the switch ack pin unconnected. This setting is useful if the implementation is a PG netlist without UPF. In this situation the corresponding black box instance the ack pin will be a black box pin and will fail when compared to an ack pin in the reference driven by the power switch from the upf applied to the reference black box.

When UPF commands are loaded onto a black box, the UPF may specify a create_power_switch with the -ack_port option. When this variable is set to 'true' , and the switch ack signal drives an output of the black box , Formality will connect ack port from the internal switch to the output of the black box. When the variable is set to 'false', Formality will leave the switch ack pin unconnected and keep the block ack output driven by the black box pin. When you have black box with an ack port and UPF in the reference and a PG netlist (without UPF) in the implementation, set the variable to false so that the ack output is a black box pin in both designs.

SEE ALSO

hdlin_interface_only(3)
load_upf(2)

upf_create_implicit_supply_sets

Controls which UPF supply nets are used for the **set_isolation -location fanout** commands.

TYPE

Boolean

DEFAULT

true

ENABLED SHELL MODES

Setup

DESCRIPTION

When loading UPF using the **set_isolation -location fanout** commands without the **-isolation_supply_set**, **-isolation_power_net** and **-isolation_ground_net** options, the default isolation supplies are used.

When this variable is set to *false* , the default isolation supplies connected because of the **set_isolation -location fanout** commands are the sink domain primary supplies.

When the variable is set to *true* (the default), the supplies for these isolation cells are the sink domain default isolation supply set instead of the sink domain primary supplies. Any explict **connect_supply_net** commands override these default connections.

SEE ALSO

load_upf(2)

upf_derive_isolation_constraints

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

This variable controls whether Formality will derive isolation constraints from the isolation strategies processed while loading UPF files. The variable must be set before issuing the **load_upf** command. The constraints generated assume each isolation signal is active when the upstream domain primary supplies turn off.

Since constraints are used as given, it is imperative that this assumption about the isolation signals be validated by simulation or other means.

SEE ALSO

load_upf(2)

upf_derive_pst_constraints

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

This variable controls whether Formality will derive design constraints from the **Power State Table (PST)** definitions processed while loading UPF files. The variable must be set before issuing the **load_upf** command.

Since constraints are assumed by Formality to be valid, its imperative that the PSTs be validated first.

SEE ALSO

load_upf(2)

upf_derive_supply_constants

Controls whether Formality derives and applies constants to supplies that are always on.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls whether the Formality tool derives and applies constants to supplies that are always on. The tool looks at the states declared by the **add_port_state** command as well the **create_power_state Power State Table (PST)** definitions processed when loading the UPF file. When it finds supplies that can only be on, the tool applies a constant to the supply net.

When this variable is set to *true*, it might improve the verification performance and completion. However, it could cause supply matching issues. This variable can only be changed when the tool is in the setup mode. As the Formality tool assumes constants to be valid, it is imperative that the PSTs be validated first.

SEE ALSO

load_upf(2)
upf_derive_pst_constraints(3)

upf_disable_spa_corruption

Specifies corruption logic introduced at lower level power domains with the **set_port_attributes** command.

TYPE

list

DEFAULT

{hier}

DESCRIPTION

This variable controls the corruption logic introduced at different hierarchical level with the UPF **set_port_attributes** command.

Any combination of the following values can be specified for the upf_disable_spa_corruption list value argument:

- **none** - all SPA corruption is allowed
- **top** - disable SPA corruption for top-level ports; hierarchical, black-box and macro ports will continue to see SPA corruption buffers
- **hier** - disable SPA corruption for hierarchical ports; top-level, black-box and macro ports will continue to see SPA corruption buffers
- **design** - disable SPA corruption for all top-level and hierarchical ports; black-box and macro ports will continue to see SPA corruption buffers
- **bbox** - disable SPA corruption for black-box and macro ports; top-level and hierarchical ports will continue to see SPA corruption buffers

EXAMPLES

```
fm_shell (setup)> set upf_disable_spa_corruption {top hier}
top hier
fm_shell (setup)> set upf_disable_spa_corruption design
design
fm_shell (setup)>
```

SEE ALSO

load_upf(2)
upf_disable_srsn_corruption(3)

upf_disable_srsn_corruption

Specifies corruption logic introduced at lower level power domains with the **set_related_supply_net** command.

TYPE

list

DEFAULT

{none}

DESCRIPTION

This variable controls the corruption logic introduced at different hierarchical level with the UPF **set_related_supply_net** command.

Any combination of the following values can be specified for the upf_disable_srsn_corruption list value argument:

- **none** - all SRSN corruption is allowed
- **top** - disable SRSN corruption fortop-level ports; hierarchical, black-box and macro ports will continue to see SRSN corruption buffers
- **hier** - disable SRSN corruption for hierarchical ports;top-level, black-box and macro ports will continue to see SRSN corruption buffers
- **design** - disable SRSN corruption for alltop-level and hierarchical ports; black-box and macro ports will continue to see SRSN corruption buffers
- **bbox** - disable SRSN corruption forblack-box and macro ports; top-level and hierarchical ports will continue to see SRSN corruption buffers

EXAMPLES

```
fm_shell (setup)> set upf_disable_srsn_corruption {top hier}
top hier
fm_shell (setup)> set upf_disable_srsn_corruption design
design
fm_shell (setup)>
```

SEE ALSO

load_upf(2)
upf_disable_spa_corruption(3)

upf_enable_state_propagation_in_add_power_state

Controls whether Formality will propagate supply set states to the individual supply set function supplies.

TYPE

String

DEFAULT

"unset"

ENABLED SHELL MODES

Setup

DESCRIPTION

Legal variable values are "true", "false", and "unset".

The variable value "true" causes Formality to continue to propagate supply set states to the individual component supply set supplies based on their corresponding terms in the supply set equation. This has been the default behaviour since "add_power_state" was first supported so that the states could be used in PSTs.

The variable value "false" enables a new semantic such that the states are only properties of the supply set as a whole and cannot be referenced in a PST although they still can be referenced in a GROUP power state. In this new mode all the states defined for a supply set are considered the only legal states allowed for the supply set and will constrain the verification accordingly.

The variable value "unset" will be same as "false" except sub-hierarchies of the design can use the old semantic via the setting of the design attribute "enable_state_propagation_in_add_power_state" in the UPF as in the following example.

```
set_design_attributes -elements {mid_inst} \
  -attribute enable_state_propagation_in_add_power_state TRUE
```

SEE ALSO

load_upf(2)

upf_enforce_strict_name_checks

Specifies that the **load_upf** command performs additional checks for object name conflicts.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable prevents naming conflicts between UPF created objects, UPF keywords, and existing design objects.

Set the **upf_enforce_strict_name_checks** variable to **true** (the default) to report errors when creating UPF objects if it detects a conflict with existing objects or UPF keywords.

If the UPF file loaded without error in earlier versions of Formality, but now has errors due to these checks, set the **upf_enforce_strict_name_checks** variable to **false** before running the **load_upf** command to turn off the checks.

EXAMPLES

The following UPF reports an error because of object name conflicts.

```
create_supply_port ss1
create_supply_set ss1
```

SEE ALSO

load_upf(2)

upf_error_on_logic_net_reconnect

Enables an error check when applying the UPF, if 'connect_logic_net' commands try to reconnect a pin

TYPE

Boolean

DEFAULT

false

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to *true*, the tool issues an error if any of the applied 'connect_logic_net' commands try to reconnect a pin from a previous logic net to a new one.

When the variable is set to *false* (default), the 'connect_logic_net' commands are free to use the option '-reconnect'.

SEE ALSO

load_upf(2)
connect_logic_net(2)

upf_ground_logic_value

TYPE

integer

DEFAULT

1

DESCRIPTION

This variable controls what value Formality uses as the logical value of the ON state for UPF ground nets when loading UPF files. The variable must be set before issuing the **load_upf** command. By default, Formality uses 1 as the logical value of the ON state for all UPF supply nets, including ground nets, as specified by the UPF standard. The other legal value is 0. If 0 is used, certain UPF features such as the use of the same UPF supply net for both power and ground connections, and connections between UPF supply nets and pins of undefined supply types, are not available. Formality inverts ground nets as necessary for UPF-to-non-UPF ground connections.

SEE ALSO

load_upf(2)

upf_hetero_fanout_isolation

Controls if Formality will do path based isolation of ports with heterogeneous fanout.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Per the LRM, -sink or -diff_supply_only strategies only apply to ports that have loads with homogenous related supplies. When this variable is set to true, Synopsys tools will group the fanout of ports with heterogeneous sinks into different paths of homogenous sinks. For each path, the most appropriate isolation strategy will be applied.

SEE ALSO

load_upf(2)

upf_implementation_based_on_file_headers

TYPE

Boolean

DEFAULT

true

DESCRIPTION

This variable controls whether Formality implements UPF constructs based on the file headers or not.

When the variable is set to **true**, Formality reads the comment at the beginning of the UPF file to determine which tool created the UPF file. Design Compiler and IC Compiler UPF designs are assumed to have different constructs implemented in the netlist, and Formality reads the header to determine the UPF constructs that need to be implemeneted.

When this variable is set to **false**, the value of the **upf_implemented_constructs** variable is used to determine which constructs in the UPF files have already been implemented in the current design.

SEE ALSO

upf_implemented_constructs(3)
load_upf(2)

upf_implemented_constructs

TYPE

list

DEFAULT

{}

DESCRIPTION

This variable determines the UPF constructs that Formality assumes are already implemented in the design being processed when the UPF file is loaded using the **load_upf** command.

Use this variable only when the **upf_implementation_based_on_file_headers** variable is set to **false**.

The value is a string of the following names which correspond UPF constructs:

```
Value               Corresponding UPF commands
-----             --------------------------
repeater            set_repeater
isolation           set_isolation/set_isolation_control
retention           set_retention/set_retention_control
power_switch        create_power_switch
supplies            create_supply_port/create_supply_net/connect_supply_net

partial_power_switch  use when switches are only partialy implemented
partial_supplies      use when supplies are only partialy implemented
```

Formality will not implement the constructs specified in the list. The default value of an empty list means that all constructs will be implemented by default.

This variable may be set before running the **load_upf** command, or inside the UPF file, which allows better control of the specific parts of the UPF that should not be implemented.

EXAMPLE

The following example shows how to use the variable to prevent implementation of set_repeater, set_isolation, and set_retention strategies when the the UPF file top.mapped.upf is applied to the implementation design.

```
fm_shell> set upf_implementation_based_on_file_headers false
fm_shell> set upf_implemented_constructs {repeater isolation retention}
fm_shell> load_upf -i top.mapped.upf
```

SEE ALSO

load_upf(2)
upf_implementation_based_on_file_headers(3)

upf_infer_clamp_for_retention

Specifies to implement clamps and zero pin retention for all retention strategies defined in the **load_upf** command.

TYPE

boolean

DEFAULT

false

DESCRIPTION

When this variable is set to true Formality will use clamps and infer zero pin retention during **load_upf** instead of retention containing save/restore pins. When **upf_use_library_cells_for_retention** is also set to true Formality will derive the clamps from attribute information found on the zero pin library cell. **upf_infer_clamp_for_retention** is intended to be used primarily with **upf_use_library_cells_for_retention** set to true. If **upf_use_library_cells_for_retention** set to false, a non-standard single DFF zero pin retention model is implemented and clamps are based on the UPF. Please contact Synopsys if you have a need for this non-standard usage.

SEE ALSO

upf_use_library_cells_for_retention(3)
load_upf(2)

upf_infer_complex_retention_cells

Specifies to use complex retention library cells in place of pre-instantiated complex non-retention cells during **load_upf**. The complex retention library cells are found in UPF command **map_retention_cell -lib_cells{}** for retention during the **load_upf** command.

TYPE

boolean

DEFAULT

false

DESCRIPTION

upf_infer_complex_retention_cells is to be used in conjunction with **upf_use_library_cells_for_retention**. When **upf_infer_complex_retention_cells** is set to false (the default), tool will not infer any complex retention library cells in place of pre-instantiated complex non-retention library cells. The pre-instantiated complex non-retention cells will be left as is and cause an error when contained within a retention strategy since Formality will be unable to retain those registers.

When this variable is set to true, Formality will infer complex retention library cells in place of pre-instaniated complex non-retention sequential library cells having retention strategies. A synchronizer is an example of a complex non-retention library cell.

Formality will try to infer a complex retention library cell during **load_upf** based on pin name matching. Here are the steps that will be performed:

1. Pin names of the non-retention library cell will be matched with the pin names of the retention library cells specified in the UPF command map_retention_cell -lib_cells{}.

2. The first retention library cell in the lib_cells {} list whose pin names exactly match all of the non-retention library cell pins will be chosen and swapped in place of the non-retention cell. All the pins present on the non-retention library cell must also be present on the retention library cell. The retention library cell may have additional test pins (in addition to the retention save/restore pins). Examples of additional pins are scan input and scan enable pins. When a retention library cell having additional pins is inferred by Formality, then the additional input pins will be connected to constant zero.

3. If a matching retention library cell is not found, then an error message will be printed saying that the non-retention cell cannot be matched to a retention cell.

4. If a **retention_equivalent** attribute is found on the non-retention library cell this will take priority over the default pin name matching in steps 1-3. If a match cannot be made using the retention_equivalent attribute then **load_upf** will default back to steps 1-3. There are two usage models in setting the retention_equivalent attribute:

Usage 1 - Exact pin name match

Use the named retention cell in the retention_equivalent attribute to do an exact pin name mapping.

set_cell_type non-retention_cell_path -value retention_equivalent -map {retention_cell_name}

Example:
fm_shell (setup)> **set_cell_type** -value retention_equivalent *r:/TCB_SYN/SYNCH_CELL* -map {RVSN_RETENTION}

RETENTION_EQUIVALENT is set on r:/TCB_SYN/SYNCH_CELL
1

Usage 2 - Pin names don't match

Map the named retention cell in the retention_equivalent attribute using the pin mapping information.
a. All pins must be mapped except for save/restore and pg
b. To invert the sense of any pin, the SystemVerilog bit-wise negation operator ~ can be specified before the retention cell pin name.
c. Extra pins are allowed such as test but must be mapped with a constant **1** or **0**.

set_cell_type non-retention_cell_path -value retention_equivalent -map {retention_cell_name {list of pin name mapping pairs}}

Example:
fm_shell (setup)> **set_cell_type** -value retention_equivalent *r:/TCB_SYN/SYNCH_CELL* -map {RVSN_RETENTION {{CP CP} {D D} {CD ~CDN} {0 SE} {0 SI} {
RETENTION_EQUIVALENT is set on r:/TCB_SYN/SYNCH_CELL
1

SEE ALSO

upf_use_library_cells_for_retention(3)
set_cell_type(2)
load_upf(2)

upf_iso_filter_elements_with_applies_to

Controls the filtering behavior when you specify the **-applies** to and **-elements** options of the **set_isolation** and \set_level_shifter commands.

TYPE

string

DEFAULT

"ENABLE"

DESCRIPTION

When you specify the **-applies** to and **-elements** options of the **set_isolation** and \set_level_shifter commands, the tool can filter the design elements such as ports, pins, and design instances. The **upf_iso_filter_elements_with_applies_to** variable controls the filtering behavior. The valid values are *ERROR*, *ENABLE*, and *DISABLE*.

- *ERROR* (default): Generates an error message when you specify the **-applies_to** option with the **-elements** option.
- *ENABLE*: Filters the elements, pins, port, and design instances based on the value that you specify with the **-applies_to** option.
- *DISABLE*: Ignores the **-applies_to** option and applies the isolation strategy to the elements specified using the **-elements** option. The isolation strategy is also applied to the pins of the design instance specified using the **-elements** option.

SEE ALSO

set_isolation(2)
set_level_shifter(2)

upf_isols_allow_instances_in_elements

Specifies the type of bias pg_pin (supply pin) to use with the **connect_supply_net** command.

TYPE

Boolean

DEFAULT

true

DESCRIPTION

When **true** instances are allowed in the element list of **set_isolation** in addition to ports/pins.

SEE ALSO

load_upf(2)

upf_keep_power_model_feedthroughs

This variable controls whether Formality preserves feedthrough paths for matched output ports when generating Formality Power Models.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

When this variable is true Formality will preserve feedthrough paths in Formality Power Models (FPM). Feedthrough paths may consist of buffer/inverter chains. If Formaity finds some complex cells/designs other than buffer/inverter chains driving an output port it is not considered a feedthrough path. When there is a difference in feedthroughs on matched output ports between reference and implementation designs, when the FPMs are used during verification of the parent design differences in feedthroughs may cause faling points. Set this variable to false to disable feedthroughs for matched output ports. Feedthrough paths on unmatched output ports are not affected by this variable.

EXAMPLE

fm_shell> set **upf_keep_power_model_feedthroughs** false

SEE ALSO

write_power_model(2)
read_power_model(2)

upf_match_dc_find_objects_transitive_behavior

TYPE

Boolean

DEFAULT

false

DESCRIPTION

Historically, with UPF command 'find_objects -transitive TRUE', DC treats the slash '/' as an embedded character, NOT as a hierarchy separator. However, the UPF LRM specifies find_objects treat '/' as hierarchy separator in this case. As a result, when this variable is set to *true*, find_objects would conform to DC's behavior; otherwise it would follow the LRM (by default).

Note that nowadays most other Synopsys tools (including Fusion Compiler) follow the LRM. DC is the only notable exception.

The variable must be set before issuing the **load_upf** command.

SEE ALSO

load_upf(2)

upf_name_map

TYPE

list of list of strings

DEFAULT

{}

DESCRIPTION

This variable specifies the name map files Formality should used in the **load_upf -strict_check false** command. It overrides any **upf_name_map** pragmas in the target design.

Its value is a list-of-list of strings where each sublist must be a {design_name map_file} pair of two elements. The design_name specifies the name of the target design; the map_file is the map file for that design. The empty string "" can be specified as a map file name if the design has an empty map file content, i.e. only default matching rules should be applied for that design.

Formality searches for name map files using $search_path, then the directory where the on-disk target design file originated.

EXAMPLE

fm_shell> **set upf_name_map [list {top top.map} {mid mid.map} {bot ""}]**

SEE ALSO

load_upf(2)

upf_power_model_library

TYPE

list

DEFAULT

""

DESCRIPTION

This variable is used to specify a list of file names that contain UPF power model definitions. Formality will load these files at the beginning of the **load_upf** command and before reading the specified top level UPF file. The power model definitions read from the files will be available in the top level scope (and it decendants) during UPF processing. The files will be found by looking in the directories specified by the tool **search_path** variable. In the Synopsys UPF flow, any hard_macro implicit power model definitions must be defined in the **upf_power_model_library** files.

EXAMPLE

fm_shell> set **upf_power_model_library** { power_model1.upf model2.upf more_models.upf }

SEE ALSO

upf_power_model_search_path(3)
search_path(3)
load_upf(2)

upf_power_model_search_path

TYPE

list

DEFAULT

""

DESCRIPTION

This variable specifies a list of directories containing UPF power model files. During **load_upf**, when the UPF **apply_power_model** *power_model_name* command is encountered and the *power_model_name* has not been defined, the tool will search for a file name of the format *power_model_name.upf* in the directories specified by this variable. If a file is found, Formality will load the file and use the power model definition. Formality will only use this variable to search for files when the *power_model_name* is not yet defined in the UPF.

EXAMPLE

fm_shell> set **upf_power_model_search_path** { power_model_directory_path another_directory_path2 }

SEE ALSO

upf_power_model_library(3)
load_upf(2)

upf_report_inferred_isolation

Specifies that the **load_upf** command should generate reports about the isolation cells that are inferred by Formality when the **load_upf** command is run.

TYPE

Boolean

DEFAULT

false

DESCRIPTION

When the variable is set to **true**, Formality generates the inferred isolation report when loading the UPF file. Formality generates a report, in .txt file format, for each container in which UPF files are loaded. The name of the directory in which the report is saved is formality_upf.

EXAMPLES

After you run the **load_upf** command on both the default reference and implementation containers, the working directory contains the following files:

```
% ls formality_upf
isolation.i.rpt    isolation.r.rpt
```

The following shows a example report.

```
#Generated by Formality (H-2013.03-Beta2) on Mon Jan 21 11:05:06 2013

set formality_iso_list {
  { /add2/sum[0] /PD_adder2 strategy1 /add2/sum[0] L 1 }
  { /add2/sum[1] /PD_adder2 strategy1 /add2/sum[1] L 1 }
  { /add2/sum[2] /PD_adder2 strategy1 /add2/sum[2] L 1 }
  { /add2/sum[3] /PD_adder2 strategy1 /add2/sum[3] L 1 }
  { /add2/sum[4] /PD_adder2 strategy1 /add2/sum[4] L 1 }
  { /add2/sum[5] /PD_adder2 strategy1 /add2/sum[5] L 1 }
  { /add2/sum[6] /PD_adder2 strategy1 /add2/sum[6] L 1 }
  { /add2/sum[7] /PD_adder2 strategy1 /add2/sum[7] L 1 }
}
```

SEE ALSO

load_upf(2)
report_upf(2)

upf_suppress_message_in_etm

Suppress message for dis-allowed command on ETM scope.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

When loading UPF at scope of an ETM instance, only *create_power_domain, add_port_state* and *set_scope* are allowed. If the UPF for the ETM contains other commands those commands will be ignored. When this variable is set to true, no warning messages are given for otherwise valid skipped commands. When this variable is set to false, warning messages will be given for the skipped commands.

SEE ALSO

load_upf(2)

upf_suppress_mv_cells_on_primary_port_with_pad

Suppress power intent from being inserted between primary ports and pad cells.

TYPE

Boolean

DEFAULT

"true"

DESCRIPTION

Since physically only a bonding wire exists between the top level port and a pad cell, no isolation or repeater cells will be inserted on such nets. Also the related supply is determined by the pad as well therefore any set_related_supply_set or set_port_attribute -driver_supply/-receiver_supply commands for ports on these nets are also ignored.

Set the variable to false to disable the special handling of primary ports with pads.

SEE ALSO

load_upf(2)

upf_unconnected_bias_pins_on

This variable controls how undriven bias pins of technology cells are handled when the bias flow is not enabled.

TYPE

Boolean

DEFAULT

true

ENABLED SHELL MODES

Setup

DESCRIPTION

You must use the same setting for the **upf_unconnected_bias_pins_on** variable, as used in the simulator to simulate the RTL and UPF together using the /fB-power/fP option.

For example, during simulation if you use the **-power=unconnected_bias_pins_on** option, then in Formality, set the following variable as follows:

set_app_var upf_unconnected_bias_pins_on true

This variable has the following effects:

- When the bias flow is not enabled and this variable is set to *false*, undriven bias pins are automatically connected to the appropriate power and ground of the parent domain's primary supply set.
- When the bias flow is not enabled and this variable is set to *true*, undriven bias pins are automatically connected to always-on constant supplies. This setting solves issues with always-on buffered paths through a shutdown domain.
- When the bias flow is enabled that results in ignoring this variable, undriven bias pins are automatically connected to the appropriate nwell and pwell functions of the parent domain's primary supply set.

SEE ALSO

load_upf(2)

upf_use_additional_db_attributes

TYPE

Boolean

DEFAULT

false

DESCRIPTION

This variable controls how Formality uses DB library attributes to govern UPF interpretation.

When this variable is true:

- If a cell is marked as a clock gating cell, a retention_cell or an isolation_cell containing a sequential element (latch based isolation cell) then the cell will not be retained regardless of the UPF set_retention strategies. It is an error if the tool encounters any instantiated technology library cell (with a `celldefine definition) register that is subject to set_retention, and no DB model is available for that cell.
- For DB macro cells (is_macro_cell : true) Formality will use the macro cell pin related_power/related_ground attributes to determine the related supplies when implementing UPF set_isolation -source/-sink/-diff_supply_only.

When this variable is false:

- Formality will issue a warning and will convert all sequential cells in a retention domain to retention cells, and it will ignore the related_power/related_ground attributes on macro cells when inserting isolation.

In a Synopsys flow, if the **synopsys_auto_setup** variable is set to *true*, the **upf_use_additional_db_attributes** variable is automatically set to *true*.

SEE ALSO

load_upf(2)
synopsys_auto_setup(3)

upf_use_instantiated_isolation_tech_cells

Specifies to search for and use isolation library cells found in the design before implementing isolation elements found in the UPF specified isolation strategies.

TYPE

boolean

DEFAULT

false

DESCRIPTION

When this variable is set to true Formality will search for pre-instantiated isolation library cells in the design before implementing one. If a library cell is found and it passes checks based on the current UPF isolation strategy then Formality will suppress implementation for that strategy element.

SEE ALSO

load_upf(2)

upf_use_library_cells_for_retention

Specifies to use library cells found in UPF command **map_retention_cell -lib_cells{} -lib_cell_type {}** for retention during the **load_upf** command.

TYPE

boolean

DEFAULT

false

DESCRIPTION

When this variable is set to true Formality will not use the default balloon latch for inferring retention during **load_upf**. Formality will map library cells found in the UPF command **map_retention_cell -lib_cells{} -lib_cell_type {}** for retention. When **upf_use_library_cells_for_retention** is set to true Formality cannot gaurantee retention behavior matches simulation.

SEE ALSO

load_upf(2)

upf_warn_on_failed_parallel_resolved_check

Changes the error message that Formality issues when there are multiple drivers on a parallel resolved supply net that cannot be determined to have the same root supply voltage into a warning message.

TYPE

Boolean

DEFAULT

true

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to *false*, in accordance with the IEEE-1801 specification, the tool issues an error when loading the UPF file if it finds a parallel resolved supply net which has multiple drivers and it cannot determine if the root supply voltage of the drivers is the same.

When the variable is set to *true*, the error message is reduced to a warning so that the **load_upf** command can proceed.

SEE ALSO

load_upf(2)

upf_warn_on_failed_port_attribute_check

Changes the error message that Formality issues when there are intermediate hierarchical ports with set_related_supply_net supplies or UPF port_attributes that do not match the actual drivers/recievers supplies to a warning message.

TYPE

Boolean

DEFAULT

false

ENABLED SHELL MODES

setup

DESCRIPTION

When this variable is set to *false* (default), the tool issues an error when loading a UPF file if during source/sink traversal it finds a hierarchical port/pin with a set_related_supply_net or UPF port_attribute -receiver_supply/-driver_supply that does not match the actual receiver/driver. This is necessary in a hiearchical flow to prevent possible invalid verification results for when the child block has been previously verified but its port attribute is in conflict with the actual driver/reciever supplies when it is instantiated in the parent design.

When the variable is set to *true*, the error message is reduced to a warning so that the**load_upf** command can proceed.

SEE ALSO

load_upf(2)

upf_warn_on_failed_unresolved_check

Changes the error message that Formality issues when there are multiple drivers on a unresolved supply net into a warning message.

TYPE

Boolean

DEFAULT

false

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to *false*, in accordance with the IEEE-1801 specification, the tool issues an error when loading the UPF file if it finds a unresolved supply net which has multiple drivers.

When the variable is set to *true*, the error message is reduced to a warning so that the **load_upf** command can proceed. The net will be treated as parallel resolved.

SEE ALSO

load_upf(2)

upf_warn_on_missing_csn_object

Changes the error message that Formality issues when there are missing ports in the specified list of a **connect_supply_net** command. The error will be reduced into a warning.

TYPE

Boolean

DEFAULT

false

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to *false* (default), in accordance with theIEEE-1801 specification, the tool issues an error when loading the UPF file if it cannot find a port from the specified list of a **connect_supply_net** command.

When the variable is set to *true*, the error message is reduced to a warning so that the**load_upf** command can proceed.

SEE ALSO

load_upf(2)
connect_supply_net(2)

upf_warn_on_missing_name_map

Changes the error message that Formality issues when using the Golden UPF flow and the DC netlist is missing the name map pragma. The error will be reduced into a warning.

TYPE

Boolean

DEFAULT

false

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to *false* (default), the tool issues an error when loading the UPF file using the Golden UPF flow if it cannot find name map pragma in the DC netlist as this indicates the Golden UPF flow was not used in DC.

When the variable is set to *true*, the error message is reduced to a warning so that the**load_upf** command can proceed.

SEE ALSO

load_upf(2)

upf_warn_on_missing_retention_element

Changes the error message to a warning message when the Formality tool cannot find an element in the UPF command 'set_retention -elements <element_list>' during **load_upf**.

TYPE

Boolean

DEFAULT

false

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to *false*, the tool issues an error message during **load_upf** when it encounters a missing element from the UPF command 'set_retention -elements <element_list>'.

When this variable is set to *true*, it changes the error message to a warning message and the**load_upf** command can complete successfully.

SEE ALSO

load_upf(2)

upf_warn_on_retention_mapping_save_restore_conflict

Specifies to change error **FM_UPF-621** to a warning.

TYPE

boolean

DEFAULT

false

DESCRIPTION

When this variable is set to true Formality will warn instead of error for **FM_UPF-621**. This is not advised since the specified UPF save/restore behavior may not match the save/restore behavior of the library cell.

SEE ALSO

upf_use_library_cells_for_retention(3)
FM_UPF-621(n)

upf_warn_on_undriven_backup_pgpin

Changes the error message to a warning message when the Formality tool find an unconnected backup PG pin during auto supply connection.

TYPE

Boolean

DEFAULT

false

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to *false*, the tool issues an error message during the post-processing step of auto supply connection when the tool encounters a backup PG pin that has not already connected to a supply by using explicitly the **connect_supply_net** command.

When this variable is set to *true*, it changes the error message to a warning message for completing the post-processing step. The backup PG pin remains undriven.

SEE ALSO

load_upf(2)

upf_warn_on_unmapped_retention_cells

Changes an error message to a warning message when the Formality tool cannot find a suitable replacement retention cell from the UPF command 'map_retention_cell -lib_cells <library_cell_list>' for an unread sequential element during**load_upf**.

TYPE

Boolean

DEFAULT

false

ENABLED SHELL MODES

Setup

DESCRIPTION

When **upf_use_library_cells_for_retention** is set to true Formality replaces sequential cells with library cells using the UPF command 'map_retention_cell -lib_cells <library_cell_list>'. The tool issues an error message during**load_upf** when it encounters a sequential element which does not have a suitable retention cell replacement from the <library_cell_list>. If the sequential element is unread then a warning will be issued instead of an error when **upf_warn_on_unmapped_retention_cells** is set to true.

SEE ALSO

upf_use_library_cells_for_retention(3)
load_upf(2)

upf_write_power_model_output_isolation_supply_only

TYPE

Boolean

DEFAULT

"false"

DESCRIPTION

The **upf_write_power_model_output_isolation_supply_only** variable controls related supplies that are preserved for ports when creating a Formality Power Model (FPM) using the **write_power_model** command.

When this variable is set to false (the default), all the supplies on the ports of the design are preserved in the FPM. When this variable is set to true, and there are isolation cells connected to the top-level output ports of the design, only the output supply of the isolation cell that drives the output is preserved. By setting the **upf_write_power_model_output_isolation_supply_only** variable to true, you make the tool assume that the isolation supply is the only supply that affects the port. This is an implicit assumption that the isolation cells are always isolating when their data input is powered off and other supplies do not affect those ports.

In practice, if you set the **upf_derive_isolation_constraints** variable to true, the mentioned assumption is used during verification. Under such circumstances, it is consistent with the verification (and recommended) to set the **upf_write_power_model_output_isolation_supply_only** variable to true when generating the FPM.

SEE ALSO

load_upf(2)
upf_derive_isolation_constraints(3)

verification_allow_hardware_x_semantics

Controls whether Formality uses "hardware", as opposed to simulation, semantics for X propagation.

TYPE

String

DEFAULT

"off"

ENABLED SHELL MODES

Setup

DESCRIPTION

Legal variable values are "off", "r2r", "g2g", "r2g" and "auto".

The variable value "off" causes Formality to treat each source of an unknown value as as an X value that is propagated consistently with simulation semantics. For example, "AND(NETA, NOT(NETA))" is X when NETA is X, though in real hardware the result would be 0 regardless of whether NETA were 0 or 1. This can cause verification failures that would also appear in simulation, but would not appear in real hardware.

The variable value "r2r" has the same effect as the variable value "off".

The variable value "g2g" causes Formality to treat every net that is a potential source of an unknown value as a matchable object, except for undriven nets, whose treatment is controlled by the variable verification_set_undriven_signals. Unknown-source nets are multiply-driven nets, nets driven by TRI cells, and nets driven by DC cells. These nets are not compare points so they do not appear in compare point status reports, but they are input points, so they may appear in matched/unmatched point reports. Their type is designated as "Unk."

The variable value "r2g" applies the traditional simulation X interpretation to the reference and the hardware X interpretation to the implementation.

The variable value "auto" determines heuristically whether the reference and implementation designs appear to be RTL or gate-level, and applies R2R, R2G or G2G accordingly.

"g2g" must be used for safety whenever "r2g" or "g2g" was previously used to verify the current reference design as an implementation design in Formality. "auto" or "g2g" must be used for safety whenever "auto" was previously used to verify the current reference design as an implementation design.

verification_alternate_strategy

Specifies that verification uses a nonstandard strategy for solving *hard* verifications.

TYPE

string

DEFAULT

none

DESCRIPTION

Use this variable to specify an alternate strategy to run verification. The standard strategy is optimized for performance and completion, but sometimes a hard verification does not complete due to design complexity. In this case, using an alternate strategy might complete verification successfully. The default is *none*, which uses the standard strategy to run verification.

See the **verification_alternate_strategy_names** variable for a list of names of all alternate strategies and for the recommended order of using an alternate strategy.

Some strategies are effective only if they are used from setup mode. The s5, s6, k1, and k2 strategies must be used only in setup mode. If the Formality tool is not in setup mode, these strategies issue an error message, and the value set for the **verification_alternate_strategy** variable is not considered. In this case, use the **setup** command to revert to setup mode before setting the **verification_alternate_strategy** variable.

The example files of UNIX Bourne shell scripts are provided at the following path to simplify using alternate strategies: $SYNOPSYS/auxx/fm/strategy/<strategy_name>.sh. These scripts also revert to setup mode for those strategies that require being used in setup mode. To get the usage information, use the UNIX Bourne shell script without arguments.

The following C shell script can be customized for different flows and environments to use alternate strategies and run these alternate strategies either in series or in a compute resource.

```
#!/bin/csh
$SYNOPSYS/bin/fm_shell -x \
  'echo $verification_alternate_strategy_names > \
  fm_verification_alternate_strategy_names.txt; quit' >& /dev/null
foreach strategy_name (`cat fm_verification_alternate_strategy_names.txt`)
  if ("none" != ${strategy_name}) then
    set strategy_script = $SYNOPSYS/auxx/fm/strategy/${strategy_name}.sh
    set dir = ${strategy_name} # Example; modify as desired.
    mkdir ${dir}
    cd ${dir}
    <Launch "${strategy_script} -s my_formality_session.fss -f my_formality_script.fms",
     potentially using lsf or qsub, etc.>
    cd ..
  endif
end
```

EXAMPLES

```
fm_shell (setup)> get_app_var verification_alternate_strategy
none
fm_shell (setup)> set_app_var verification_alternate_strategy s6
s6
fm_shell (setup)>
```

SEE ALSO

verification_alternate_strategy_names(3)

verification_alternate_strategy_names

Lists names of all alternate strategies. Read-only.

TYPE

list

DESCRIPTION

The read-only variable lists the names of all alternate strategies and the recommended order of using an alternate strategy. The order of the list indicates which strategy to use first. However, the most effective alternate strategy might vary from case to case.

EXAMPLE

```
fm_shell (setup)> get_app_var verification_alternate_strategy_names
none s2 s3 s1 l2 s10 s8 l1 l3 s4 s6 s5 k1 k2 s7 s9
fm_shell (setup)>
```

SEE ALSO

verification_alternate_strategy(3)

verification_assume_reg_init

Controls the assumptions that Formality makes about the initial state of registers.

TYPE

string

DEFAULT

"Auto"

ENABLED SHELL MODES

setup

DESCRIPTION

Use this variable to control the assumptions that Formality makes about the initial state of a register, which can affect whether the register resolves to a constant or not. Registers that cannot achieve any state other than their initial state are resolved to constant.

When the variable is set to "None", Formality does not make any assumptions about the initial state of registers, i.e. all registers are assumed to have an unknown initial state (X).

In "Conservative" mode, Formality assumes that registers with asynchronous controls that can be controlled to an active value by toggling primary inputs or black-box outputs are initialized according to their asynchronous set/clear capabilities. Registers with no asynchronous controls (or permanently disabled asynchronous controls) are initialized to an unknown state (X). If the only asynchronous input of a register is an asynchronous clear, the register is initialized to 0. If the only asynchronous input of a register is an asynchronous set, the register is initialized to 1. All other registers are initialized to an unknown state (X).

In "Liberal" mode, Formality assumes that registers are initialized as described in "conservative" mode. The only difference between "Conservative" and "Liberal" mode applies to potentially constant registers. Potentially constant registers are registers that always remain at the same state once they are initialized to that state. If not initialized, potentially constant registers remain at an unknown state (X) until a valid initialization event occurs. Valid initialization events are those that occur as a result of toggling one or more primary inputs or black-box outputs. In "Liberal" mode, Formality assumes that if a valid initialization event exists, it will occur at power-up. For example, a flop that has its data pin tied to 0 and its clock pin driven by a primary input is a potentially constant 0 register. This flop can be treated as constant 0 only after it is initialized (i.e. its clock is toggled). Formality assumes that this initialization event (the toggling of the clock) occurs at power-up and therefore treats the flop as constant 0.

In "Auto" mode which is a hybrid mode, "None" mode is applied to reference design while "Liberal" mode is applied to implementation design. This hybrid behaviour is specific to "Consistency" verification passing mode. In "Equality" verification passing mode, both reference and implementation designs are applied "None" register initialization mode.

In "AutoMatched" mode which is a hybrid mode, "None" mode is applied to reference design while special "matched Liberal" mode is applied to implementation design. We only initialize constant registers in the implementation if the objects controlling the control-pins

on the registers can be matched in the reference design. This hybrid behaviour is specific to "Consistency" verification passing mode. In "Equality" verification passing mode, both reference and implementation designs are applied "None" register initialization mode.

In "Liberal0" mode, Formality assumes that all registers are initialized to 0. The only exception, are registers that have (a)synchronous set capabilities and no (a)synchronous clear capabilities. Such registers are initialized to 1.

In "Liberal1" mode, Formality assumes that all registers are initialized to 1. The only exception, are registers that have (a)synchronous clear capabilities and no (a)synchronous set capabilities. Such registers are initialized to 0.

To change the value of this variable, enter **set verification_assume_reg_init** *value*, where *value* is "None", "Conservative", "Liberal" , "Liberal0" or "Liberal1".

verification_asynch_bypass

Enables or disables verification of registers with asynchronous controls operated by a combinational "bypass" around the register against registers with asynchronous controls operated by setting the register state.

TYPE

boolean

DEFAULT

"false"

DESCRIPTION

Enables or disables verification of registers with asynchronous controls operated by a combinational "bypass" around the register against registers with asynchronous controls operated by setting the register state.

Some library register components implement asynchronous control logic by creating a combinational "bypass" around the register while asynchronous (level-sensitive) controls are active. By default, these designs fail verification against those with asynchronous controls implemented purely as inputs to the register. If you want Formality to allow this transformation to result in a passing verification, set this variable to "true". Note that doing so increases verification complexity and may introduce combinational cycles.

To change the value of this variable, enter **set verification_asynch_bypass** *value*, where *value* is "true" or "false".

verification_auto_loop_break

Enables or disables automatic loop breaking.

TYPE

boolean

DEFAULT

"true"

ENABLED SHELL MODES

setup

DESCRIPTION

Use this variable to enable or disable automatic loop breaking.

Set this variable to "false" if you do not want Formality to break cycles automatically.

By default, Formality performs simple loop breaking. Exploiting structural similarities between two cycles enables Formality to automatically identify combinational cycles in the reference and implementation designs and cut feedback nets in them. Each cycle in the reference design can be matched with a cycle in the implementation design by name matching or by finding isomorphism between two cycles. Once two cycles are matched, Formality identifies and cuts nets so that the functionality of the two designs remains unchanged.

The automatic loop breaking feature is very conservative. If there is any structural difference between two cycles, Formality simply abandons automatic loop breaking and performs more powerful and complex cycle analysis.

To change the value of this variable, enter **set verification_auto_loop_break** *value*, where *value* is "true" or "false".

verification_auto_session

Specifies whether a session file is generated automatically.

TYPE

String

DEFAULT

"on"

DESCRIPTION

This variable controls automatic saving of a session file at various points during and after verification. Only the last automatically saved session file is retained.

Specify one of the following values:

- *off* - No session file is generated.
- *timeout* - A session file is generated after a verification timeout.
- *on* - A session file is generated after a verification timeout or after reaching a memory limit, or verification terminates for any reason unsuccessfully after the threshold determined by the *verification_auto_session_threshold* variable, or during distributed computing after the interval determined by the *dpx_auto_session_interval* variable.
- *always* - A session file is generated when verification finishes or times out regardless of the threshold specified with the *verification_auto_session_threshold* variable, or during distributed computing after the interval determined by the *dpx_auto_session_interval* variable.
- *verify* - A session file is generated when the value is *on* and also after each effort level of the **verify** command.
- *match* - A session file is generated when the value is *verify* and also after matching.

A session file with the name **formality_auto_session.fss** is automatically created in the directory where all other auto generated files reside (log files etc.). If a file of that name already exists when verification starts, it is named by inserting the smallest integer value (starting with one) needed to make it unique (example **formality1_auto_session.fss**).

EXAMPLE

```
fm_shell (setup)> printvar verification_auto_session
verification_auto_session = "on"
fm_shell (setup)> set verification_auto_session match
match
```

SEE ALSO

verification_auto_session_threshold(3)
dpx_auto_session_interval(3)
verification_timeout_limit(3)

verification_auto_session_threshold

Specifies a wall-clock time after which sessions are automatically saved.

TYPE

string or integer

DEFAULT

"12:0:0"

DESCRIPTION

Use this variable along with the *verification_auto_session* variable to specify the period of time that must occur before the Formality tool automatically saves a session file for a verification that does not terminate successfully. This variable takes effect only when the *verification_auto_session* variable is set to 'on', 'match' or 'verify'. Only the last automatically saved session file is retained.

The default of the *verification_auto_session_threshold* variable is 12 hours. You can use positive integers for hours or the hours:minutes:seconds format.

To change the value of this variable, use **set verification_auto_session_threshold** *value*, where *value* is a positive integer for hours or is in the hours:minutes:seconds format.

For example, to indicate a 30-minute threshold, specify 0:30:00. If you specify 30, Formality interprets it as 30:0:0; that is, a 30-hour threshold.

SEE ALSO

verification_auto_session(3)

verification_binary_constraint_propagation

Controls the method that Formality uses to find constant and constrained registers.

TYPE

boolean

DEFAULT

"false"

ENABLED SHELL MODES

setup

DESCRIPTION

When this variable is true, Formality uses a new method to determine constrained registers during verification. The new algorithm never assumes that a register is initialized to the X value. If it does not receive an initial value of 0 or 1 through the setting of verification_assume_reg_init, then the initial state will be 01, i.e., binary. Registers can be found to be constrained to 0X or 1X only if the X comes from a don't-care cell in the netlist.

SEE ALSO

verification_assume_reg_init(3)

verification_blackbox_match_mode

Defines how Formality matches comparable black boxes during verification.

TYPE

String

DEFAULT

"any"

ENABLED SHELL MODES

Setup

DESCRIPTION

Use this variable to define how Formality matches comparable black boxes during verification.

By default, Formality matches two comparable black boxes regardless of the library in which they reside and regardless of their design name. However, you can tighten the control on this behavior by setting this variable to "identity". This setting causes Formality to perform an identity check between comparable black boxes. During this check, Formality determines if the library and design names of the two black boxes are identical. If so, the black boxes pass the check and are considered equivalent during verification. If the black boxes fail the check, Formality does not consider the two black boxes as equivalent. (Note that when there is a set user match between two black boxes, Formality will match them regardless of the value of verification_blackbox_match_mode)

For information on how to gain more control of how Formality controls black boxes, refer to "Working with Black Boxes" in the user guide.

To change the value of this variable, enter **set verification_blackbox_match_mode** *value*, where *value* is "any" or "identity".

verification_clock_gate_edge_analysis

Specifies whether the Formality tool uses clock edges in the next state formulation of registers or not. This allows the tool to identify clock gating latches and circuitry during verification.

TYPE

Boolean

DEFAULT

"false"

ENABLED SHELL MODES

Setup

DESCRIPTION

Clock gating is a design technique that reduces the power consumption of registers in a design. When you set the variable to *true*, the tool uses clock edges for analyzing clock gated designs. This allows the tool to verify designs with different styles of clock gating.

When you set the **verification_clock_gate_edge_analysis** variable to **true**, the tool

- Ignores any usage of the **verification_clock_gate_hold_mode** variable.
- Adds annotations to clock signals indicating their present state and next state values. The annotations are visible in the pattern viewer and logic cone schematics.

You might see the following annotations when analyzing failing compare points:

Annotation	Present State	Next State
r (rising edge)	0	1
f (falling edge)	1	0
0->X	0	X
1->X	1	X
X->0	X	0
X->1	X	1

SEE ALSO

verification_clock_gate_hold_mode(3)

verification_clock_gate_reverse_gating(3)

verification_clock_gate_hold_mode

Specifies the mode in which the Formality tool should identify and treat clock gates driving register clock pins.

TYPE

string

DEFAULT

"none"

ENABLED SHELL MODES

setup

DESCRIPTION

The **verification_clock_gate_hold_mode** variable allows the Formality tool to apply algorithms for identifying and treating latch-free and latch-based clock gates that drive register clock pins.

By default, the tool does not apply the algorithms, and the verification might result with a difference. To change the value, use the **set verification_clock_gate_hold_mode** *value* variable. Where *value* is one of the following:

- none (the default): Does not apply clock-gate algorithms.
- low: Considers latch-based clock gating ("latch-and" driving rising edge and "latch-or" driving falling edge) and latch-free clock gating, where the gated clock is held at a value consistent with the value before the edge ("*en&clk*" driving rising edge and "*!en|clk*" driving falling edge)
- high: Considers latch-free clock gating, where the gated clock is held at a value consistent with the value after the edge ("*en&clk*" driving falling edge and "*!en|clk*" driving rising edge)
- any: Support for this option is discontinued starting with S-2021.06. Recommendation is to use verification_clock_gate_reverse_gating.
- collapse_all_cg_cells: Same as *low*, but also considers all primary output ports and black-box input pins as register clock pins.

SEE ALSO

verification_clock_gate_edge_analysis(3)
verification_clock_gate_reverse_gating(3)

verification_clock_gate_reverse_gating

Specifies whether the Formality tool identifies and reverses synthesis tools' insertion of clock gating latches.

TYPE

Boolean

DEFAULT

"false"

ENABLED SHELL MODES

Setup

DESCRIPTION

Clock gating is a design technique that reduces the power consumption of registers in a design. When you set the variable to *true*, the tool identifies and reverses the insertion of clock gating latches. Internally, the Formality tool will remove the clock gating latch, and connect the clock and enable pins directly to the receiving registers' clock (CLK) and synchronous enable (SL) pins. This transformation does not modify the functional circuit behavior, but allows the tool to verify clock gated designs.

When you set the **verification_clock_gate_reverse_gating** variable to **true**, the tool

- Ignores any usage of the **verification_clock_gate_hold_mode** and **verification_clock_gate_edge_analysis** variables.
- Identifies and reverses clock gating latches in the design.

SEE ALSO

verification_clock_gate_hold_mode(3)
verification_clock_gate_edge_analysis(3)

verification_constant_prop_mode

Controls where Formality starts constant propagation from, during verification.

TYPE

string

DEFAULT

"auto"

ENABLED SHELL MODES

setup

DESCRIPTION

Use this variable to specify where Formality starts constant propagation from, during verification. Formality propagates all constants (both user-defined and design constants) through all levels of hierarchy.

In "top" mode, Formality propagates constants starting from the top level reference and implementation designs. In "target" mode, Formality propagates constants starting from the currently set reference and implementation designs. In "auto" mode, Formality automatically determines the appropriate level of hierarchy to start constant propagation from. This is done by traversing up the current reference and implementation hierarchy until a unique instance is found.

Design constants are nets in a design tied to a logical 0 or 1 state. User-defined constants are created using the set_constant command.

To change the value of this variable, enter **set verification_constant_prop_mode** *value*, where *value* is one of the following: "top", "auto" or "target".

verification_datapath_effort_level

Defines the effort level Formality applies during datapath block verification.

TYPE

string

DEFAULT

"automatic"

ENABLED SHELL MODES

setup

DESCRIPTION

Use the verification_datapath_effort_level variable to set the effort level Formality applies during datapath block verification. Effort levels include low, medium, high, unlimited, and automatic (the default).

During verification of designs with datapath blocks in them, Formality first tries to pre-verify these arithmetic blocks. In the case of successful pre-verification, the verified datapath block gets black boxed for the rest of the verification, which can significantly shorten the entire verification time. By default, Formality automatically sets a proper effort level to either low, medium, or high for each datapath block verification by estimating difficulty of the datapath block. (For an explanation of the various levels see the definitions that follow.) If the pre-verification is inconclusive with the default setting, you can increase the chance of a conclusivepre-verification by setting the effort level explicitly to a non-automatic value, but at a potential cost to the run time of thepre-verification. However, an inconclusive pre-verification does not necessarily mean a longer overall verification run time. There could be cases where the main verification flow might solve the verification of the entire design quicker by setting the effort level for pre-verification to "low".

Note that with the "unlimited" effort level it is possible that the pre-verification might continue for a considerable amount of time.

low : Limit the resources to finish datapath pre-verification in seconds.

medium: Limit the resources to finish datapath pre-verification in minutes.

high : Limit the resources to finish datapath pre-verification in hours.

unlimited: No resource limit.

automatic: An effort level for each datapath block is automatically determined to either low, medium, or high (default).

verification_effort_level

This variable controls how hard Formality works to verify a set of compare points.

TYPE

string

DEFAULT

"High"

ENABLED SHELL MODES

all

DESCRIPTION

Use this variable to control how hard the **verify** command works to verify a set of compare points before aborting the unsolved points and going on to the next set of points. This variable is useful for hard or long-running verifications if you verify incrementally. First set this variable to **Super_Low** or **Low** or **Medium**, run **verify**, examine the results to see the big picture of where the problems might be, and then if some points remain unsolved, set the variable to **High** to attempt to verify the complex parts of the design.

The default value of this variable is "High".

To change the value of this variable, enter **set verification_effort_level** *value*, where *value* is "High", "Medium", "Low" or "Super_Low".

SEE ALSO

verify(2)

verification_failing_pattern_limit

Specifies the number of failing patterns saved for each failing compare point.

TYPE

integer

DEFAULT

8

DESCRIPTION

Use this variable to set the maximum number of failing patterns that Formality saves for each failing compare point.

You must provide a positive integer. The default value is 8. Setting this variable to 0 results in Formality saving all failing patterns for all failing points.

If there are many failing points and many patterns for each failing point, then multicore runs and session files can require large amounts of disk space. You can limit the number of patterns using this variable to reduce the disk usage.

verification_failing_point_limit

Specifies the number of failing compare points identified before Formality halts the verification process.

TYPE

integer

DEFAULT

20

DESCRIPTION

Use this variable to set the maximum number of failing compare points that Formality allows before halting verification.

You must provide a positive integer. The default value is 20 (does not include compare points comprising a single design object). For unlimited failing compare points, you can supply a value of "zero" (0).

For conceptual information about failing compare points, refer to "Compare Points" in the user guide.

To change the value of this variable, enter **set verification_failing_point_limit** *value*, where *value* is an integer.

verification_force_upf_supplies_on

Controls whether Formality verifies all possible UPF power states, or only the UPF power state where all supplies are on.

TYPE

String

DEFAULT

"true"

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is true, Formality will verify the UPF based designs in the state where all supplies are forced into an "on" state. Formality will disable all power state table information supplied in the UPF and apply constant values to the UPF supply nets to force them into an on state.

IMPORTANT: This is not a complete verification for a UPF design. For a complete verification you must run verification with this variable set to false. It is also is possible that the state where all supplies are on, is not a legal UPF power state. This can only be determined by examining the power state table in the upf files.

When this variable is false Formality will verify all possible power states as defined by the UPF files. This is a complete low power verification of the design and should to make sure all legal combinations of power states is verified. Formality should be run in this mode before the implementation design is used as the reference for subsequent verifications.

This variable can only be changed in setup mode.

Because the load_upf command is required to identify the appropriate supply nets, this variable should be kept at true only when load_upf has been used for both the reference and implementation designs. It may cause an unsuccessful verification when a UPF design is being verified is a power and ground connected netlist.

This variable will be ignored and treated as false if the variable 'verification_verify_power_off_states' is set to true.

SEE ALSO

load_upf(2)

verification_verify_power_off_states(3)

verification_ignore_unmatched_always_on_pg_pins

Defines whether Formality allows unmatched unread pg pins on always_on techlib cells in a succeeding verification.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to controlthe effect on the verification result of unmatched unread *primary_power* and *primary_ground* input pins on techlib cells having the *always_on* db attribute.

"true" [default]: unmatched unread pg pins on always_on cells in the reference design will not cause verification failure.

"false": the unmatched unread primary pg pins will cause verification failure.

To change the value of this variable, enter **set fBverification_ignore_unmatched_always_on_pg_pins** *value*, where *value* is "true" or "false".

verification_ignore_unmatched_implementation_blackbox_input

Defines whether Formality allows unmatched implementation blackbox input pins in a succeeding verification.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to control the effect of unmatched implementation blackbox input pins on the verification result.

"true" [default]: unmatched blackbox input pins in the implementation design will not cause verification failure, if the matching reference blackbox has no unmatched input pins.

"false": any unmatched blackbox input pin will cause verification failure.

To change the value of this variable, enter **set verification_ignore_unmatched_blackbox_input** *value*, where *value* is "true" or "false".

verification_ignore_unmatched_implementation_output_port

Defines whether Formality allows unmatched implementation output ports in a succeeding verification.

TYPE

boolean

DEFAULT

"true"

DESCRIPTION

Use this variable to control the effect of unmatched implementation output ports on the verification result.

"true" [default]: unmatched output ports in the implementation design will not cause verification failure, if the reference has no unmatched output ports.

"false": any unmatched output ports will cause verification failure.

To change the value of this variable, enter **set verification_ignore_unmatched_implementation_output_port** *value*, where *value* is "true" or "false".

verification_incremental_mode

This variable controls whether the verify command verifies incrementally.

TYPE

string

DEFAULT

"true"

ENABLED SHELL MODES

all

DESCRIPTION

Use this variable to control the default behavior of the **verify** command. Normally, if you issue the **verify** command more than once, it retains the status of previously verified compare points and attempts only to verify points that were previously aborted due to limits. You can control whether or not **verify** re-verifies previously verified compare points using the **verify -restart** and **verify -incremental** switches. If you do not use these switches, then **verify** depends on the value of the **verification_incremental_mode** variable. If **verification_incremental_mode** is **true**, verify behaves incrementally. If **verification_incremental_mode** is **false**, verify re-verifies all points, including those that have been previously verified, if any.

The default value of this variable is "true".

To change the value of this variable, enter **set verification_incremental_mode** *value*, where *value* is "true" or "false".

SEE ALSO

verify(2)

verification_insert_upf_clock_cutpoints

This variable controls whether Formality inserts power down cutpoints (PDCut) on domain boundaries which are also clock signals.

TYPE

Boolean

DEFAULT

"false"

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to *true*, the tool inserts cutpoints on power domain boundaries that are part of clock lines.

The default value of this variable is false because inserting cutpoints at power domain boundaries which are part of clock lines can cause verification failing points when there are clock line/clock gating differences between the two designs. In cases where there are no changes in the clock gating logic it should be safe to set this variable true, and it can help prevent failing points due to X propagation differences on the clock line.

This variable can only be changed when the tool is in the setup mode.

SEE ALSO

verification_insert_upf_isolation_cutpoints(3)
verification_insert_upf_macro_cutpoints(3)
load_upf(2)
set_dont_cut(2)

verification_insert_upf_isolation_cutpoints

This variable controls whether Formality inserts cutpoints at isolated power domain boundaries.

TYPE

Boolean

DEFAULT

"true"

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to *true*, the tool inserts cutpoints at isolated UPF power domain boundaries.

Inserting cutpoints at power domain boundaries improves verification time and eliminates failures caused by simulation-only differences in X propagation when unisolated power-off states propagate across a power domain boundary. However, in some cases it may also cause false differences, for example when boundary optimization has occurred across an isolation cell.

This variable can only be changed when the tool is in the setup mode.

SEE ALSO

verification_insert_upf_clock_cutpoints(3)
verification_insert_upf_macro_cutpoints(3)
load_upf(2)
set_dont_cut(2)

verification_insert_upf_logic_expr_cutpoints

Directs Formality in ALL-ON mode to insert cutpoints at the inputs of UPF logic expressions of supply set power states.

TYPE

String

DEFAULT

false

ENABLED SHELL MODES

setup

DESCRIPTION

In ALL-ON mode where supplies are forced to their ON state, the controls to switches will be effectively un-read by the switches. As such they will not be verified if there are no other readers leading to compare points. If the switched supply is used in a supply set and the power switch controls are used in the logic expression of the power states of that supply set, this variable can close that verification hole.

By setting **verification_insert_upf_logic_expr_cutpoints** to *true*, when **verification_force_upf_supplies_on** is *true*, Formality will put cutpoints on the input signals to the logic expressions. These cuts become compare points to verify the power switch controls.

SEE ALSO

verification_force_upf_supplies_on(3)

verification_insert_upf_macro_cutpoints

This variable controls whether Formality inserts cutpoints at macro cell boundaries, including 4 modes {*none*, *input*, *output*, *both*}.

TYPE

String

DEFAULT

"none"

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is set to anything other than *none*, and UPF has been loaded, the tool inserts cutpoints at the macro cell boundary.

There are four choices for cutpoint insertion:

- **none** - the default, no cutpoints are inserted at macro cell boundaries
- **input** - cutpoints are inserted at macro cell input pins
- **output** - cutpoints are inserted at macro cell output pins
- **both** - cutpoints are inserted at both macro cell input and output pins

Cutpoints are never inserted at bidirectional(inout) macro cell pins.

For backward compatibility the previous value of *false* is the same as *none*, while *true* is the same as *both*.

Inserting cutpoints at macro cell boundaries may improve verification time and eliminate failures caused by simulation-only differences in X propagation when unisolated power-off states propagate across a macro cell boundary. However, in some cases it may also cause differences, for example when buffer optimization has occurred across a macro cell boundary. In these cases, use the **set_dont_cut** command to prevent cutpoint insertion on the pins of the macro that are causing the failures.

This variable can only be changed when the tool is in the setup mode.

SEE ALSO

verification_insert_upf_isolation_cutpoints(3)
verification_insert_upf_clock_cutpoints(3)
load_upf(2)
set_dont_cut(2)

verification_inversion_push

Controls whether Formality's matching methods attempt to account for cases where data inversion has been moved across register boundaries. This variable is available only in library verification mode.

TYPE

boolean

DEFAULT

"false"

ENABLED SHELL MODES

library_setup

DESCRIPTION

Use this variable to control whether Formality's matching methods attempt to account for cases where data inversion has been moved across register boundaries.

Note: This variable is available only in library verification mode. In normal design verification it is obsolete and has been removed from Formality starting in the S-2021.06 release. In design verification mode, it is recommended to use SVF. SVF has guide_inv_push commands indicating which registers have phase inversions.

verification_merge_duplicated_registers

Specifies whether Formality should identify and merge duplicated registers.

TYPE

string

DEFAULT

"false"

ENABLED SHELL MODES

setup

DESCRIPTION

Use this variable to control whether Formality identifies and merges duplicated registers in all designs under verification.

Duplicated registers are registers in the same design whose input pins are all driven by the same nets. If this variable is set to "true", Formality will identify such registers and collapse them into a single register. This can allow verification to succeed when a design contains duplicate registers but the design to which you are comparing it does not.

By default, Formality does not identify and merge duplicated registers.

You can change the default behavior by setting this variable "true".

To change the value of this variable, enter **set verification_merge_duplicated_registers** *value*, where *value* is "true" or "false".

verification_merge_parallel_switches

DESCRIPTION

This variable has been renamed to **hdlin_merge_parallel_switches**.

SEE ALSO

hdlin_merge_parallel_switches(3)

verification_netlist_verify_mode

This variable controls if Formality attempts a first pass of verification targetted for fast netlist compares.

TYPE

string

DEFAULT

"Auto"

ENABLED SHELL MODES

all

DESCRIPTION

Use this variable to enable fast verification of netlist compares in Formality.

Allowed values are "Off", "On" or "Auto".

When this mode is enabled, Formality tries a fast round of verification without performing expensive operations like constant propagation. The default value of this variable is "Auto". "Auto" mode tries to automatically detect if verification is a netlist compare.

To change the value of this variable, enter **set verification_netlist_verify_mode** *value*, where *value* is "Off", "On" or "Auto".

verification_parameter_checking

This variable enables parameter checking for technology library register cells and black-box cells.

TYPE

string

DEFAULT

"false"

ENABLED SHELL MODES

setup

DESCRIPTION

By setting this variable to true, users can verify that user-defined or default parameters on matched technology library register cells and matched black box cells are maintained throughout the design flow. User-defined parameters are those specified with defparam statements on technology library register cells as well as black box cells. Default parameters are those defined in your technology library cells.

By default, Formality does not perform parameter checking.

You can change the default behavior by setting this variable "true".

To change the value of this variable, enter **set verification_parameter_checking** *value*, where *value* is "true" or "false".

verification_partition_timeout_limit

Specifies wall time limit for verification of a single partition.

TYPE

string or integer

DEFAULT

"0"

ENABLED SHELL MODES

all

DESCRIPTION

Use this variable to specify maximum time allowed for verification of each partition.

Normally, Formality continues to perform verification until it either proves or disproves design equivalence. However, if you want to limit the time used for verification of each partition, you can use this variable to set a maximum wall-clock time limit applied to each partition. Formality halts the verification of the partition when reaching the limit regardless of the state of the verification, and proceeds to the next partition, if any.

You must enter positive integers for hours and minutes. By default, Formality places no time limit on verification of a partition. Use "none" to return the time limit to its default.

To change the value of this variable, enter **set verification_partition_timeout_limit** *value*, where *value* is an integer or in hours, minutes, and/or seconds using the following format: hours:minutes:seconds.

For example, to indicate a 60 second time limit, specify 0:0:60. If you specify 60, Formality interprets it as 60:0:0; that is, a 60 hour time limit.

verification_passing_mode

Specifies the verification mode.

TYPE

string

DEFAULT

"Consistency"

DESCRIPTION

Use this variable to control the type of verification Formality performs.

By default, Formality checks for design consistency. However, you can also instruct Formality to base verification on design equality. For information that describes the difference between these two types of equivalence testing, refer to "Design Equivalence" in the user guide. Once you set the verification mode, that mode remains in effect for the entire Formality session.

To change the value of this variable, enter **set verification_passing_mode** *value*, where *value* is "Consistency", or "Equality".

verification_progress_report_interval

Specifies the interval between progress reports during verification, in minutes.

TYPE

integer

DEFAULT

30

DESCRIPTION

Use this variable to specify how much time elapses between each progress report during verification. By default, during long verifications Formality issues a progress report every 30 minutes. For more or less frequent updates, use this variable to set the interval between progress reports.

To change the value of this variable, enter **set verification_progress_report_interval** *value*, where *value* is the desired interval in minutes. Setting the variable to 0 disables verification progress reports.

verification_propagate_const_reg_x

Specifies how Formality propagates don't-care states through reconvergent-fanout-dependent constant-disabled registers.

TYPE

string

DEFAULT

"false"

DESCRIPTION

When a register in the reference design is enabled or clocked by logic that resolves to a constant, and the constant puts the register into a permanently disabled state, no possible input pattern can cause it to load data. Therefore its next-state function is always a don't-care, and is never verified.

In some cases, downstream logic in the implementation design may have been optimized under the assumption that the register's state is always a don't-care. Therefore, if the constant that disables the register does not arise from reconvergent fanout, Formality automatically treats the register as a constant X input to downstream logic. However, if the constant stems from reconvergent fanout, Formality takes a more conservative view, because the constant-disabled state may be unexpected and arise from the presence of timing-dependent functionality in the register's control line. In this case Formality issues a warning, and by default does not propagate the X state past the register. In which case you may encounter downstream compare point failures whose failing patterns are controlled by a 0/1 state at the disabled register. If the constant-disabled state is expected, you can set this variable "true." This will cause the register to be viewed as a constant X during verification of downstream compare points.

To change the value of this variable, enter **set verification_propagate_const_reg_x** *value*, where *value* may be "true" or "false."

verification_run_analyze_points

This variable controls whether Formality automatically runs **analyze_points** at the end of verification.

TYPE

boolean

DEFAULT

"false"

ENABLED SHELL MODES

setup

DESCRIPTION

When this variable is true Formality will automatically run **analyze_points** prior to returning from verification and will include a brief summary of the results in the transcript. The full results will be added to the formality log file and can also be displayed in the transcript by running the **report_analysis_results** command.

SEE ALSO

analyze_points(2)
report_analysis_results(2)

verification_set_constant_no_corrupt

Controls whether Formality propagates all user **set_constant** values through power-down function don't care cells without causing corruption.

TYPE

boolean

DEFAULT

"false"

ENABLED SHELL MODES

setup

DESCRIPTION

When this variable is true, Formality propagates all user set_constant values through power-down functions without X corruption. This variable only takes effect when the variable **verification_binary_constraint_propagation** is true. This can be used to allow constants set by the user to disable scan logic or other unverifiable differences to propagate to the compare point registers during all-state verification.

SEE ALSO

verification_binary_constraint_propagation(3)
set_constant(2)
verification_force_upf_supplies_on(3)

verification_set_undriven_signals

Specifies how Formality treats undriven nets and pins during verification.

TYPE

String

DEFAULT

"BINARY:X"

ENABLED SHELL MODES

Setup

DESCRIPTION

Use this variable to control the value that Formality assumes for undriven nets and pins.

By default, Formality treats undriven pins and nets in the reference design as BINARY (cut points) and in the implementation design as X. This conservative value results in verification failure if any matched compare point in the reference design is ever controlled by any undriven signal. This ensures that your reference design's behavior is not controlled by any unexpected undriven signals. To be sure your reference and implementation designs propagate undriven net or pin values to compare points identically, you need to specify the value BINARY for this variable.

Note: When the **synopsys_auto_setup** Tcl variable is set to *true*, Formality sets the **verification_set_undriven_signals** variable to *SYNTHESIS*.

To change the value of the variable, use the **set verification_set_undriven_signals** *value* variable. Where *value* is one of the following:

- *X*: Support for this option is discontinued starting withS-2021.06. Treats undriven pins and nets as X (don't-care in the reference design and don't-know in the implementation design to be consistent with the simulation).
- *Z*: Support for this option is discontinued starting withS-2021.06. Treats undriven pins and nets as Z (high-impedance).
- *0*: Treats undriven pins and nets as 0.
- *1*: Treats undriven pins and nets as 1.
- *0:X*: Treats undriven pins and nets in the reference design as 0 and in the implementation design as X (ensures undriven reference signals are tied to 0 in implementation).
- *BINARY*: Treats each undriven pin or net as a matchable independent free variable by creating a cut point at each undriven signal. This value results in verification failure when the downstream compare points are controlled by unmatched (undriven

signal) cut points.

- *BINARY:X*: The default. Treats undriven pins and nets in the reference design as BINARY and in the implementation design as X. This value results in verification failure if any matching reference compare point is controlled by any undriven signal.
- *SYNTHESIS*: Treats reference undriven pins and nets as treated by the Design Compiler tool and implementation undriven pins and nets as BINARY.
- *SYNTH:SYNTH*: Treats reference and implementation undriven pins and nets as treated by the Design Compiler tool. This can not be set, and is used internally during reference to reference compares.

The PI value for this variable is deprecated and support discontinued starting with S-2021.06.

SEE ALSO

synopsys_auto_setup(3)

verification_static_low_power_compare

Enables static low power comparison checks between the reference and implementation designs and UPF.

TYPE

String

DEFAULT

"false"

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is true, during the **match** command Formality will invoke VCLP to process the reference and implementation designs and UPF and during the **verify** command Formality will automatically run the **compare_lp** command to perform static low power comparison checks between the reference and implementation. Any differences between the two designs will be flagged with UPF_DIFF* violations in the **compare_lp** results report.

IMPORTANT NOTES:

- The Linux environment variable VC_STATIC_HOME must be set to a valid VC_STATIC (VCLP) installation tree before this variable is set to true.
- This variable must be set before any design data is read so that the designs can be processed correctly for low power comparison.
- The static low power comparision will not work when reading only Formality containers because the design source files for all designs read into Formality must be available so that the files read can be read by the background VCLP sessions that are run by Formality.

SEE ALSO

compare_lp(2)
report_lp(2)
set_vclp_setup_commands(2)
vclp_start_gui(2)

verification_status

Returns the status of the most recent verification, if any.

TYPE

string

DEFAULT

"NOT RUN"

DESCRIPTION

Use this variable to see the status of the most recent verification. Possible values are "SUCCEEDED", "FAILED", "INCONCLUSIVE", "MATCHED", "NOT RUN", "GUIDE".

You cannot change the value of this variable.

verification_timeout_limit

Specifies a wall-clock time limit for verification commands.

TYPE

string or integer

DEFAULT

"36:0:0"

DESCRIPTION

Use this variable to specify maximum wall-clock time allowed for the **preverify**, **match**, and **verify** commands.

The **preverify**, **match**, and **verify** commands either run to completion or stop when the value specified by **verification_timeout_limit** (default is 36 hours) is reached. If the time limit specified is reached, the tool will interrupt the current state of verification.

You must enter positive integers for hours and minutes. To perform verification until design equivalence is either proven or disproven, specify *none*.

To change the value of this variable, enter **set verification_timeout_limit** *value*, where *value* is an integer or in hours, minutes, and/or seconds using the following format: hours:minutes:seconds.

For example, to indicate a 60 second time limit, specify 0:0:60. If you specify 60, Formality interprets it as 60:0:0; that is, a 60 hour time limit.

SEE ALSO

preverify(2)
match(2)
verify(2)

verification_verify_directly_undriven_output

Verifies directly-undriven output ports.

TYPE

Boolean

DEFAULT

"true"

ENABLED SHELL MODES

Setup

DESCRIPTION

This variable verifies the directly undriven ports.

Setting this variable to **false** to ignore (not verify) the directly-undriven output ports. In the synthesis or verification flow, directly-undriven output ports are typically created for the insertion of scan test circuitry later in the flow.

By default, Formality verifies directly-undriven output ports, which typically fails verification when the **verification_set_undriven_signals** variable is to the default.

To avoid such false failures, set the **verification_verify_directly_undriven_output** variable to *false*. In a Synopsys flow, if the **synopsys_auto_setup** variable is set to *true*, the **verification_verify_directly_undriven_output** variable is automatically set to *false*.

SEE ALSO

set_dont_verify_points(2)
synopsys_auto_setup(3)

verification_verify_matched_unread_bbox_inputs

Controls if matched unread black-box input pins are verified by Formality.

TYPE

String

DEFAULT

"true"

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is "true", Formality will verify input pins of unread black-box cells. This mode can identify differences when verifying modules that have digital inputs but only produce analog outputs and hence will not have any output pins visible to Formality.

When this variable is "false", black-boxes with no output pins will be marked as unread and Formality will not verify the input pins because such compare points cannot affect the functionality of the outputs of a design.

This variable is ignored when variables **verification_verify_unread_bbox_inputs** or **verification_verify_unread_compare_points** is true.

SEE ALSO

verification_verify_unread_bbox_inputs(3)
verification_verify_unread_compare_points(3)
verification_verify_unread_tech_cell_pins(3)

verification_verify_matched_unread_compare_points

Allows only matched unread compare points to be verified by the Formality tool.

TYPE

string

DEFAULT

"false"

ENABLED SHELL MODES

setup

DESCRIPTION

You can verify only matched unread compare points when this variable is set to *true*. Unread compare points are registers or black_box inputs where

- those points do not have other compare points in their fanout

  Or

- those points fanout to only other unread compare points.

By default, the Formality tool does not verify unread compare points because such compare points cannot affect the functionality of the outputs of a design. This variable is ignored when variable **verification_verify_unread_compare_points** is *true*.

SEE ALSO

verification_verify_unread_compare_points(3)
verification_verify_unread_bbox_inputs(3)
verification_verify_unread_tech_cell_pins(3)

verification_verify_unread_bbox_inputs

Controls if matched and unmatched unread black-box input pins are matched and verified by Formality.

TYPE

String

DEFAULT

"false"

ENABLED SHELL MODES

Setup

DESCRIPTION

When this variable is "true", Formality will verify input pins of unread black-box cells. This mode can identify differences when verifying modules that have digital inputs but only produce analog outputs and hence will not have any output pins visible to Formality.

When this variable is "false", black-boxes with no output pins will be marked as unread and Formality will not verify the input pins because such compare points cannot affect the functionality of the outputs of a design.

This variable is ignored when variable **verification_verify_unread_compare_points** is true.

SEE ALSO

verification_verify_unread_compare_points(3)
verification_verify_unread_tech_cell_pins(3)
verification_verify_matched_unread_bbox_inputs(3)

verification_verify_unread_compare_points

Allows unread compare points for match and verify by the Formality tool.

TYPE

string

DEFAULT

"false"

ENABLED SHELL MODES

setup

DESCRIPTION

You can verify all unread compare points when the variable is set to *true*. Unread compare points are registers or black_box inputs where

- those points do not have other compare points in their fanout

  Or

- those points fanout to only other unread compare points.

By default, the Formality tool does not verify unread compare points because such compare points cannot affect the functionality of the outputs of a design.

SEE ALSO

verification_verify_matched_unread_compare_points(3)
verification_verify_unread_bbox_inputs(3)
verification_verify_unread_tech_cell_pins(3)

verification_verify_unread_tech_cell_pg_pins

This variable is used to tell Formality to verify or ignore unread technology library cell power/ground (PG) pins. It has no effect when variable **verification_verify_unread_tech_cell_pins** is set false.

TYPE

string

DEFAULT

"true"

ENABLED SHELL MODES

setup

DESCRIPTION

When **verification_verify_unread_tech_cell_pins** is set to true, then this variable is used to control the verification of unread technology library cell PG pins. This variable has no effect when **verification_verify_unread_tech_cell_pins** is set to false.

When **verification_verify_unread_tech_cell_pins** is set to true, and this variable is true (the default) uread PG pins will be verified. When **verification_verify_unread_tech_cell_pins** is set to true, and this variable is set to false, Formality will not verify unread PG pins.

By setting this variable to false the logic driving unread tech cell input PG pins will be considered unread and will not be verified. Be careful with this setting, as part of the design may be excluded from verification.

This variable must be set before the set_top command.

SEE ALSO

verification_verify_unread_tech_cell_pins(3)

verification_verify_unread_tech_cell_pins

This feature allows unread input pins of tech library cells in the reference design to be matched with unread input pins of tech library cells in the implementation design, and treated as compare points for verification. Unmatched unread input pins of tech library cells in either design will cause a verification failure.

TYPE

string

DEFAULT

"true"

ENABLED SHELL MODES

setup

DESCRIPTION

During the set_top command processing, if Formality finds a tech cell input pin that has no loads inside of the tech cell, that pin is considered to be an unread tech cell pin. For each such pin, Formality will create a black-box cell with one input pin inside of the tech cell and connect it to the unread pin. The name of each new black-box cell will be derived from that of the input pin of tech library cell, and will also include the text "unread". The effect of this design modification will be that no unread input pins of tech library cells will remain in both designs. The unread input pins of tech library cells will be read by individual black-box input pins.

By setting this variable to false, all logic driving unread tech cell input pins will be considered unread and will not be verified. Be careful with this setting, as part of the design may be excluded from verification.

Unmatched unread input pins of tech library cells in either design will cause a verification failure.

SEE ALSO

verification_verify_unread_tech_cell_pg_pins(3)

verification_verify_upf_supplies

Controls whether Formality verifies UPF supply nets as distinct compare points.

TYPE

String

DEFAULT

"auto"

ENABLED SHELL MODES

setup

DESCRIPTION

By default (*auto*), Formality verifies UPF supply nets when **verification_force_upf_supplies_on** is *false* and the supply nets are matched by name. This variable allows you to override the default and control whether UPF supply nets become compare points, by setting it to *true* or *false*.

If **verification_verify_upf_supplies** is *true* and **verification_force_upf_supplies_on** is *true*, Formality forces the readers of supply nets to their on values, but not the supply nets themselves, and the supply nets become compare points.

SEE ALSO

verification_force_upf_supplies_on(3)
verification_insert_upf_isolation_cutpoints(3)