ESP
Variables

Version O-2018.06, June 2018

SYNOPSYS®

Copyright Notice and Proprietary Information

© 2018 Synopsys, Inc. All rights reserved. This software and documentation contain confidential and proprietary information that is the property of Synopsys, Inc. The software and documentation are furnished under a license agreement and may be used or copied only in accordance with the terms of the license agreement. No part of the software and documentation may be reproduced, transmitted, or translated, in any form or by any means, electronic, mechanical, manual, optical, or otherwise, without prior written permission of Synopsys, Inc., or as expressly provided by the license agreement.

Destination Control Statement

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

Disclaimer

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

Trademarks

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

Free and Open-Source Licensing Notices

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

Third-Party Links

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

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

Contents

bigendian

NAME

bigendian

Force all bus declarations to be bigendian.

TYPE

Boolean

Legal values: *on off*

DEFAULT

off

DESCRIPTION

Force all bus declarations to be bigendian.

This should only be used for compatibility with older versions of the software.

SEE ALSO

Commands: **read_spice**

Variables: **netlist_aggressive_net_compression** **netlist_aggressive_port_compression** **netlist_bus_extracti**

collection_result_display_limit

NAME

collection_result_display_limit

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

TYPE

int

Legal values: -1 to 2,147,483,647

DEFAULT

100

DESCRIPTION

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

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

To determine the current value of this variable, use:

```
printvar  collection_result_display_limit
```

SEE ALSO

Commands: **collections** **printvar** **query_objects**

coverage

NAME

coverage

Enable coverage data gathering during verification.

TYPE

Boolean

Legal values: *true false*

DEFAULT

false

DESCRIPTION

When true, coverage data is gathered during verification.

SEE ALSO

Commands: **report_coverage**

filter_collection_extended_syntax

TYPE

boolean

DEFAULT

application specific

DESCRIPTION

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

SEE ALSO

`filter_collection(2)`

flattening_device_count

NAME

flattening_device_count

Limits the number of devices to be flattened.

TYPE

integer

Legal values: 0 to 2,147,483,647

DEFAULT

2,500,000

DESCRIPTION

Limits the number of devices to be flattened when searching for weak devices.

Note: This is for pure switch level simulation only. Do not use in the default RC mode.

SEE ALSO

Commands: read_spice report_design_spice_mode reset_design_spice_mode set_design_spice_mode set_instan

match_port_range_strict

NAME

match_ports_strict

Top level port match control.

TYPE

string

Legal values: *true false*

DEFAULT

true

DESCRIPTION

This variable controls how strictly the top level netlist ports are matched. When the variable value is *true*, all top level ports must match by name, buses must contain the same bit numbers and range and there must be the same number of signals between the top level designs being compared.

SEE ALSO

Commands: match_design_ports remove_matched_ports report_matched_ports report_unmatched_ports set_matc

match_ports_strict

NAME

match_ports_strict

Top level port match control.

TYPE

string

Legal values: *true false*

DEFAULT

true

DESCRIPTION

This variable controls how strictly the top level netlist ports are matched. When the variable value is *true*, all top level ports must match by name, buses must contain the same bit numbers and range and there must be the same number of signals between the top level designs being compared.

SEE ALSO

Commands: match_design_ports remove_matched_ports report_matched_ports report_unmatched_ports set_matc

mos_transconductance_ratio

NAME

mos_transconductance_ratio

Strength ratio between PFET and NFET type devices.

TYPE

floating

Legal values: >0.0

DEFAULT

2.00

DESCRIPTION

Defines the strength (also called mobility) ratio between an NFET and PFET for default process devices.

SEE ALSO

Commands: read_spice report_design_spice_mode report_process reset_design_spice_mode set_design_spice_

Other: **device_model_simulation**

netlist_aggressive_net_compression

NAME

netlist_aggressive_net_compression

Cause all subcircuit nets to be converted into Verilog vectors

TYPE

Boolean

Legal values: *on off*

DEFAULT

off

DESCRIPTION

Causes all design nets not connected to ports to be converted into Verilog vectors (bussified).

Note

Supply nets will not be converted into Verilog vectors.

Set this variable before calling the **read_spice** command.

EXAMPLE

```
set_app_var netlist_aggressive_net_compression on
```

SEE ALSO

Commands: **read_spice**

Variables: **bigendian** **netlist_aggressive_port_compression** **netlist_bus_extraction_style**

netlist_aggressive_port_compression

NAME

netlist_aggressive_port_compression

Cause all subcircuit port buses to be converted to Verilog vectors.

TYPE

Boolean

Legal values: *on off*

DEFAULT

off

DESCRIPTION

Causes all subcircuit port buses to be converted to Verilog vectors (bussification).

Note

Supply ports will not be converted into Verilog vectors.

Set this variable before calling the **read_spice** command.

EXAMPLE

```
set_app_var netlist_agressive_port_compression on
```

SEE ALSO

Commands: **read_spice**

Variables: **bigendian** **netlist_aggressive_net_compression** **netlist_bus_extraction_style**

netlist_bus_extraction_style

NAME

netlist_bus_extraction_style

Sets the naming style used to determine if a SPICE net belongs to a bus.

TYPE

string

Legal values: *%s<%d> %s[%d] %s(%d) %s_%d_ %s_%d*

Where *%s* is any string and *%d* is any positive integer. All other characters are exact matches

DEFAULT

%s<%d>

DESCRIPTION

Sets the naming style used to determine if a SPICE net or port belongs to a bus.

Only the five styles of buses shown are supported.

Set this variable before calling the **read_spice** command.

Note:
Escape the value with braces or single quotes

EXAMPLE

```
set_app_var netlist_bus_extraction_style {%s_%d_}
```

SEE ALSO

Commands: **read_spice**

Variables: **bigendian** **netlist_aggressive_net_compression** **netlist_aggressive_port_compression**

netlist_case

NAME

netlist_case

Controls case sensitivity when reading a SPICE netlist

TYPE

string

Legal values: *preserve lower*

DEFAULT

preserve

DESCRIPTION

Controls case sensitivity when reading a SPICE netlist. The default is to *preserve* case sensitivity. This is different than most SPICE simulators which behave as if there is no case difference. For most SPICE simulators the net names *addr*,*Addr* and *ADDR* are the same net. For ESP those net names are three different nets.

Set **netlist_case** to *lower* if you wish ESP to behave like most SPICE simulators.

Set this variable before calling the **read_spice** command.

EXAMPLE

```
set_app_var netlist_case lower
```

SEE ALSO

Commands: **read_spice**

netlist_inverter_chain_extraction

NAME

netlist_inverter_chain_extraction

Insert unit delays in inverter chains.

TYPE

Boolean

Legal values: *on off*

DEFAULT

off

DESCRIPTION

When *on*, chains of inverters are modified to have a unit delay added to each inverter.

If there is more than one connection on a net, the inverter will not have a unit delay inserted.

Note:
Do not set this variable to *on* when RC mode is used.

Set this variable before calling the **read_spice** command.

EXAMPLE

Enable unit delays for inverter chains when not in RC mode.

```
if {[string compare $verify_spice_simulator_mode "rcdelays"]!=0} {
  set_app_var netlist_inverter_chain_extraction on
}
```

SEE ALSO

Commands: **read_spice** **report_design_spice_mode** **reset_design_spice_mode** **set_design_spice_mode** **set_instan**

netlist_process_estimation_version

NAME

netlist_process_estimation_version

TYPE

string

Legal values: *2009.06*, *2009.12*

DEFAULT

2009.12

DESCRIPTION

Controls the default transistor models used by ESP when no SPICE technology files are provided. The default value of *2009.12* uses data based upon the model from the Arizona Predictive Technology Models.

The 2009.12 data extends down to a 7nm process node.

Note:
It is recommended that the **Device Model Simulation** methodology be used for all process nodes below 40nm. The default model has only one N device model and one P device model at each technology node. The default model does not account for various device types that often exist at each technology node.

For further information the Arizona Predictive Technology Models (PTM) , see **Predictive Technology**

Models , and the following publications:

- W. Zhao, Y. Cao, "New generation of Predictive Technology Model for sub-45nm early design exploration," IEEE Transactions on Electron Devices, vol. 53, no. 11, pp. 2816-2823, November 2006.
- Y. Cao, T. Sato, D. Sylvester, M. Orshansky, C. Hu, "New paradigm of predictive MOSFET and interconnect modeling for early circuit design," pp. 201-204, CICC, 2000.

The value of *2009.06* uses the same default values used in releases of ESP prior to the D-2009.12 release of ESP.

The use of the **set_process** command means that PTM models values are not used.

Set this variable before calling the **read_spice** command.

EXAMPLES

```
set_app_var netlist_process_estimation_version 2009.06
```

SEE ALSO

Commands: **read_spice**

Topics: **device_model_simulation**

netlist_unwrap_transistors

NAME

netlist_unwrap_transistors

Unwrap transistors during SPICE translation to Verilog.

TYPE

Boolean

Legal values: *on off*

DEFAULT

on

DESCRIPTION

Single transistor sub-circuits can have the transistor moved up a level to replace the sub-circuit instance with the actual transistor. This is called unwrapping a transistor. If there are multiple levels of single instance sub-circuits, the unwrapping will recursively occur until the transistor is no longer within a single instance sub-circuit.

With **netlist_unwrap_transistors** *off*, unwrapping of transistors is disabled.

SEE ALSO

Commands: **write_esp_db**

netlist_use_verilog_escape

NAME

netlist_use_verilog_escape

Netlist translation uses Verilog escaped identifiers.

TYPE

Boolean

Legal values: *on off*

DEFAULT

on

DESCRIPTION

Netlist translation uses Verilog escaped identifiers for identifiers that have illegal Verilog characters.

With **netlist_use_verilog_escape** *off* illegal characters are changed to _ (underscore).

SEE ALSO

Commands: **read_spice**

query_objects_format

TYPE

string

DEFAULT

application specific

DESCRIPTION

This variable sets the format that the **query_objects** command uses to print its result. There are two supported formats: Legacy and Tcl.

The Legacy format looks like this:

```
{"or1", "or2", "or3"}
```

The Tcl format looks like this:

```
{or1 or2 or3}
```

Please see the man page for **query_objects** for complete details.

SEE ALSO

query_objects(2)

sh_allow_tcl_with_set_app_var

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

TYPE

string

DEFAULT

application specific

DESCRIPTION

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

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

SEE ALSO

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

sh_allow_tcl_with_set_app_var_no_message_l

Suppresses CMD-104 messages for variables in this list.

TYPE

string

DEFAULT

application specific

DESCRIPTION

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

SEE ALSO

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

sh_arch

Indicates the system architecture of your machine.

TYPE

string

DEFAULT

platform-dependent

DESCRIPTION

The **sh_arch** variable is set by the application to indicate the system architecture of your machine. Examples of machines being used are sparcOS5, amd64, and so on. This variable is read-only.

sh_command_abbrev_mode

Sets the command abbreviation mode for interactive convenience.

TYPE

string

DEFAULT

application specific

DESCRIPTION

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

Although the default value is **Anywhere**, it is recommended that the site startup file for the application set this variable to **Command-Line-Only**. It is also possible to set the value to **None**, which disables abbreviations altogether.

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

SEE ALSO

sh_command_abbrev_options(3)
get_app_var(2)
set_app_var(2)

sh_command_abbrev_options

Turns off abbreviation of command dash option names when false.

TYPE

boolean

DEFAULT

application specific

DESCRIPTION

When command abbreviation is currently off (see sh_command_abbrev_mode) then setting this variable to false will also not allow abbreviation of command dash options. This variable also impacts abbreviation of the values specified to command options that expect values to be one of an allowed list of values.

This variable exists to be backward compatible with previous tool releases which always allowed abbreviation of command dash options and option values regardless of the command abbreviation mode.

It is recommended to set the value of this variable to false.

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

SEE ALSO

sh_command_abbrev_mode(3)
get_app_var(2)
set_app_var(2)

sh_command_log_file

Specifies the name of the file to which the application logs the commands you executed during the session.

TYPE

string

DEFAULT

empty string

DESCRIPTION

This variable specifies the name of the file to which the application logs the commands you run during the session. By default, the variable is set to an empty string, indicating that the application's default command log file name is to be be used. If a file named by the default command log file name cannot be opened (for example, if it has been set to read only access), then no logging occurs during the session.

This variable can be set at any time. If the value for the log file name is invalid, the variable is not set, and the current log file persists.

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

SEE ALSO

`get_app_var(2)`
`set_app_var(2)`

sh_continue_on_error

Allows processing to continue when errors occur during script execution with the **source** command.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

This variable is deprecated. It is recommended to use the **-continue_on_error** option to the **source** command instead of this variable because that option only applies to a single script, and not the entire application session.

When set to **true**, the **sh_continue_on_error** variable allows processing to continue when errors occur. Under normal circumstances, when executing a script with the **source** command, Tcl errors (syntax and semantic) cause the execution of the script to terminate.

When **sh_continue_on_error** is set to **false**, script execution can also terminate due to new error and warning messages based on the value of the **sh_script_stop_severity** variable.

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

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_script_stop_severity(3)
```

sh_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_page_mode

Displays long reports one page at a time (similar to the UNIX **more** command.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

This variable displays long reports one page at a time (similar to the UNIX **more** command), when set to **true**. Consult the man pages for the commands that generate reports to see if they are affected by this variable.

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

SEE ALSO

`get_app_var(2)`
`set_app_var(2)`

sh_enable_stdout_redirect

Allows the redirect command to capture output to the Tcl stdout channel.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When set to **true**, this variable allows the redirect command to capture output sent to the Tcl stdout channel. By default, the Tcl **puts** command sends its output to the stdout channel.

SEE ALSO

`get_app_var(2)`
`set_app_var(2)`

sh_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_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_product_version

Indicates the version of the application currently running.

TYPE

string

DESCRIPTION

This variable is set to the version of the application currently running. The variable is read only.

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

SEE ALSO

`get_app_var(2)`

sh_script_stop_severity

Indicates the error message severity level that would cause a script to stop running before it completes.

TYPE

string

DEFAULT

application specific

DESCRIPTION

When a script is run with the **source** command, there are several ways to get it to stop running before it completes. One is to use the **sh_script_stop_severity** variable. This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a script to stop.
- When set to **W**, the generation of one or more warning or error messages causes a script to stop.
- When set to **none**, the generation messages does not cause the script to stop.

Note that **sh_script_stop_severity** is ignored if **sh_continue_on_error** is set to **true**.

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

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)
```

sh_source_emits_line_numbers

Indicates the error message severity level that causes an informational message to be issued, listing the script name and line number where that message occurred.

TYPE

string

DEFAULT

application specific

DESCRIPTION

When a script is executed with the **source** command, error and warning messages can be emitted from any command within the script. Using the **sh_source_emits_line_numbers** variable, you can help isolate where errors and warnings are occurring.

This variable can be set to **none**, **W**, or **E**.

- When set to **E**, the generation of one or more error messages by a command causes a CMD-082 informational message to be issued when the command completes, giving the name of the script and the line number of the command.
- When set to **W**, the generation of one or more warning or error messages causes a the CMD-082 message.

The setting of **sh_script_stop_severity** affects the output of the CMD-082 message. If the setting of **sh_script_stop_severity** causes a CMD-081 message, then it takes precedence over CMD-082.

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

SEE ALSO

get_app_var(2)
set_app_var(2)
source(2)
sh_continue_on_error(3)
sh_script_stop_severity(3)
CMD-081(n)
CMD-082(n)

sh_source_logging

Indicates if individual commands from a sourced script should be logged to the command log file.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When you source a script, the **source** command is echoed to the command log file. By default, each command in the script is logged to the command log file as a comment. You can disable this logging by setting **sh_source_logging** to **false**.

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

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
```

sh_source_uses_search_path

Indicates if the **source** command uses the **search_path** variable to search for files.

TYPE

Boolean

DEFAULT

application specific

DESCRIPTION

When this variable is set to \ftrue the **source** command uses the **search_path** variable to search for files. When set to **false**, the **source** command considers its file argument literally.

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

SEE ALSO

```
get_app_var(2)
set_app_var(2)
source(2)
search_path(3)
```

sh_tcllib_app_dirname

Indicates the name of a directory where application-specific Tcl files are found.

TYPE

string

DESCRIPTION

The **sh_tcllib_app_dirname** variable is set by the application to indicate the directory where application-specific Tcl files and packages are found. This is a read-only variable.

SEE ALSO

`get_app_var(2)`

sh_user_man_path

Indicates a directory root where you can store man pages for display with the **man** command.

TYPE

list

DEFAULT

empty list

DESCRIPTION

The **sh_user_man_path** variable is used to indicate a directory root where you can store man pages for display with the **man** command. The directory structure must start with a directory named *man*. Below *man* are directories named *cat1*, *cat2*, *cat3*, and so on. The **man** command will look in these directories for files named *file.1*, *file.2*, and *file.3*, respectively. These are pre-formatted files. It is up to you to format the files. The **man** command effectively just types the file.

These man pages could be for your Tcl procedures. The combination of defining help for your Tcl procedures with the **define_proc_attributes** command, and keeping a manual page for the same procedures allows you to fully document your application extensions.

The **man** command will look in **sh_user_man_path** after first looking in application-defined paths. The user-defined paths are consulted only if no matches are found in the application-defined paths.

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

SEE ALSO

```
define_proc_attributes(2)
get_app_var(2)
man(2)
set_app_var(2)
```

spice_simulator

NAME

spice_simulator

Script to start SPICE simulator for debugging.

TYPE

script

Legal value: Bourne shell script where *%s* is replaced by a list of SPICE files

DEFAULT

hspice %s > esp_spicerun.log

DESCRIPTION

Specifies a command (or simple script) to start a SPICE simulation for debugging.

The value *%s* if present will be replaced by the list of SPICE files created by **write_spice_debug_testbench** .

This variable is used by **start_spice_simulator** to start a SPICE simulation.

SEE ALSO

Commands: **create_model_library** **export_spice_debug_testbench** **start_spice_simulator** **write_spice_debug_te**

synopsys_program_name

Indicates the name of the program currently running.

TYPE

string

DESCRIPTION

This variable is read only, and is set by the application to indicate the name of the program you are running. This is useful when writing scripts that are mostly common between some applications, but contain some differences based on the application.

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

SEE ALSO

get_app_var(2)

synopsys_root

Indicates the root directory from which the application was run.

TYPE

string

DESCRIPTION

This variable is read only, and is set by the application to indicate the root directory from which the application was run.

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

SEE ALSO

get_app_var(2)

testbench_binary_cycles

NAME

testbench_binary_cycles

Number of binary cycles used in standard testbench.

TYPE

integer

Legal values: >= 0

DEFAULT

2

DESCRIPTION

Defines the number of binary cycles in the testbench created by **write_testbench** .

If more than 10 binary cycles are needed, a custom testbench should be created instead of using the testbench created by **write_testbench** .

Note:

testbench_style is affected by this variable. Testbench styles of macro, sram, rom and cam ignore the setting of this variable.

SEE ALSO

Commands: write_testbench

Variables: testbench_constraint_file
testbench_declaration_file
testbench_dump_symbolic_waveform
testbench_flush_cycles
testbench_implementation_instance
testbench_initialization_file
testbench_module_name
testbench_output_checks
testbench_reference_instance
testbench_style
testbench_symbolic_cycles

testbench_constraint_file

NAME

testbench_constraint_file

Use a Verilog constraint file.

TYPE

string

Legal value: Any UNIX filename

DEFAULT

NONE

DESCRIPTION

Name of a Verilog constraint file to be used by all generated testbenches.

This file contains Verilog code that is legal within a Verilog task.

It will be included by all testbenches created by **write_testbench** after setting **testbench_constraint_file** to a non-null value.

The testbenches use a Verilog `include command to include the specified file within the apply_global_constraints task of the testbench.

To create a sequential constraint, **testbench_declaration_file** can be used to specify a Verilog file to define the state variables.

SEE ALSO

Commands: **write_testbench**

Variables: **testbench_binary_cycles** **testbench_declaration_file** **testbench_dump_symbolic_waveform** **testben**

testbench_declaration_file

NAME

testbench_declaration_file

Use a Verilog declaration file.

TYPE

string

Legal values: Any UNIX filename

DEFAULT

NONE

DESCRIPTION

The name of a Verilog declaration file to be used by all generated testbenches.

This file contains Verilog code that is legal in a Verilog module. One use for this file is to declare state variables for use by **testbench_constraint_file** in order to implement a sequential constraint.

It will be included by all testbenches created by **write_testbench** after setting **testbench_declaration_file** to a non-null value.

The testbenches use a Verilog `include` command to include the specified file.

SEE ALSO

Commands: **write_testbench**

Variables: **testbench_binary_cycles** **testbench_constraint_file** **testbench_dump_symbolic_waveform** **testbenc**

testbench_design_instance

NAME

testbench_design_instance

Instance name used in a custom simulation testbench to instantiate the reference or implementation design.

TYPE

string

Legal values: Any Verilog identifier

DEFAULT

ref

DESCRIPTION

Defines the instance name path used in a custom testbench to instantiate the reference or implementation design.

The **testbench_design_instance** must be set so that automatic dumping of coverage data can occur.

Failure to set **testbench_design_instance** to the correct value can result in a compilation error when coverage dumping is enabled.

SEE ALSO

Commands: **report_coverage**

Variables: **coverage**

testbench_dump_symbolic_waveform

NAME

testbench_dump_symbolic_waveform

Enables dumping of waveform data during symbolic verification.

TYPE

Boolean

Legal values: *true false*

DEFAULT

false

DESCRIPTION

Enables dumping of waveform data during symbolic verification.

Note: Nets with equations will be dumped as *X* values. Regular non-symbolic nets will have actual values dumped.

Dumping waveforms of a symbolic verification can help in finding where symbols are dropped as reported in the coverage report.

SEE ALSO

Commands: report_coverage write_testbench

Variables: coverage testbench_binary_cycles testbench_constraint_file testbench_declaration_file testb

testbench_flush_cycles

NAME

testbench_flush_cycles

Number of flush cycles used in standard testbench.

TYPE

integer

Legal values: >=0

DEFAULT

1

DESCRIPTION

Defines the number of flush cycles in the testbench created by **write_testbench** .

If more than 10 flush cycles are needed, a custom testbench should be created instead of using the testbench created by **write_testbench** .

Note:

testbench_style is affected by this variable. Testbench styles of macro, sram, rom and cam ignore the setting of this variable.

SEE ALSO

Commands: write_testbench

Variables: testbench_binary_cycles testbench_constraint_file testbench_declaration_file testbench_dump

testbench_implementation_instance

NAME

testbench_implementation_instance

Instance name used in a custom testbench to instantiate the implementation design.

TYPE

string

Legal values: any Verilog identifier

DEFAULT

ximp

DESCRIPTION

Defines the instance name used in a custom testbench to instantiate the implementation design.

If a custom testbench does not use the default *ximp* instance name to instantiate the implementation design, then **testbench_implementation_instance** must be set so that automatic dumping of coverage data and waveform data can occur.

Failure to set **testbench_implementation_instance** can result in a compilation error when waveform dumping or coverage dumping is enabled. This variable is not used or needed if **set_verify_mode** has the value *simulate*.

SEE ALSO

Commands: **set_verify_mode** **write_testbench**

Variables: **coverage** **testbench_binary_cycles** **testbench_constraint_file** **testbench_declaration_file** **testb**

Topics: **esp_shell_flow**

testbench_initialization_file

NAME

testbench_initialization_file

Use a Verilog initialization file.

TYPE

string

Legal values: Any UNIX filename

DEFAULT

NONE

DESCRIPTION

The name of a Verilog initialization file to be used by all generated testbenches.

This file contains Verilog code that is legal in a Verilog initial block. It will be included by all testbenches created by **write_testbench** after setting **testbench_initialization_file** to a non-null value.

The testbenches use a Verilog `include command to include the specified file.

Use **testbench_constraint_file** if Verilog code is intended as a constraint.

Use **testbench_declaration_file** if more general Verilog code is needed.

SEE ALSO

Commands: **write_testbench**

Variables: **testbench_binary_cycles** **testbench_constraint_file** **testbench_declaration_file** **testbench_dump**

testbench_max_vectors_in_phase

NAME

testbench_max_vectors_in_phase

Maximum number of input or inout ports that can change at the same time.

TYPE

Integer

Legal values: 0 to 2^{31}-1

DEFAULT

1

DESCRIPTION

This variable controls the maximum number of input or inout ports that can change at the same time for a library cell verification testbench.

SEE ALSO

Commands: **debug_design** **set_verification_defaults** **verify**

testbench_module_name

NAME

testbench_module_name

The top level module name used in the testbench.

TYPE

string

Legal values: Any Verilog identifier

DEFAULT

inno_tb_top

DESCRIPTION

Defines the name of the top level testbench module used in a custom testbench.

If a custom testbench does not use the default *inno_tb_top* module name as the top level testbench module then **testbench_module_name** must be set so that automatic dumping of coverage data and waveform data can occur.

Failure to set **testbench_module_name** can result in a compilation error when waveform dumping or coverage dumping is enabled.

SEE ALSO

Commands: **write_testbench**

Variables: **coverage** **testbench_binary_cycles** **testbench_constraint_file** **testbench_declaration_file** **testb**

Topics: **esp_shell_flow**

testbench_output_checks

NAME

testbench_output_checks

The number of checks per clock cycle in the testbench.

TYPE

string

Legal values: *4check 3check 1check*

DEFAULT

3check

DESCRIPTION

The number of times output is checked in the testbench per clock cycle is controlled by this variable.

Note:
The clock used for this check is the first clock specified by the **create_clock** command or the **set_testbench_pin_attributes** *-function clock* option. If no clock is specified, then ESP uses `inno_clk` as the clock.

4check applies 4 checks per clock cycle. The checks are done at SETUPTIME before the clock edge and just at the clock edge before the clock changes value. Use *4check* to get consistent checker numbers across all testbench styles.

3check applies 3 checks per clock cycle. The checks are done at SETUPTIME before the first clock edge at the start of the test cycle and just at the clock edge before the clock changes value.

1check applies 1 check per clock cycle. The check is done at the end of the cycle.

SEE ALSO

Commands: **write_testbench**

Variables: **testbench_binary_cycles** **testbench_constraint_file** **testbench_declaration_file** **testbench_dump**

testbench_reference_instance

NAME

testbench_reference_instance

Instance name used in the testbench to instantiate the reference design.

TYPE

string

Legal values: Any Verilog identifier

DEFAULT

xref

ref with the legacy parser

DESCRIPTION

Defines the instance name used in a testbench to instantiate the reference design.

If a custom testbench does not use the default instance name to instantiate the reference design, then **testbench_reference_instance** must be set so that automatic dumping of coverage data and waveform data can occur.

Failure to set **testbench_reference_instance** can result in a compilation error when waveform dumping or coverage dumping is enabled.

This variable is not used or needed if **set_verify_mode** has the value *simulate*.

SEE ALSO

Commands: **set_verify_mode** **write_testbench**

Variables: **coverage** **testbench_binary_cycles** **testbench_constraint_file** **testbench_declaration_file** **testb**

Topics: **esp_shell_flow**

testbench_setup_file

NAME

testbench_setup_file

Use a Verilog setup file.

TYPE

string

Legal values: Any UNIX filename

DEFAULT

NONE

DESCRIPTION

The name of a Verilog setup file to be used by all generated testbenches. The *setup* file is included at the top of the generated testbench immediately after the first module line.

This file contains Verilog code that is legal in a Verilog module. It will be included by all testbenches created by **write_testbench** after setting **testbench_setup_file** to a non-null value.

The testbenches use a Verilog `include command to include the specified file.

This file can be used to override everything in the generated testbenches except for the name of the top level testbench module and the timescale.

This file is the way to override the default regs, wires, defines, reference instantiation, implementation instantiation and the declaration of symbolic variables.

Use **testbench_initialization_file** if Verilog code is intended for the INIT_task.

Use **testbench_constraint_file** if Verilog code is intended as a constraint.

Use **testbench_declaration_file** if more general Verilog code is needed.

SEE ALSO

Commands: **write_testbench**

Variables: **testbench_binary_cycles** **testbench_constraint_file** **testbench_declaration_file** **testbench_dump**

testbench_style

NAME

testbench_style

Identify the testbench style to be generated for use in verification.

TYPE

enumerated string

Legal values: *cam cammatch clkenum clkwave dataint dualphs holdchk library macro portcov protocol rom romhold romproto sram symbolic*

DEFAULT

symbolic

DESCRIPTION

The style of the testbench generated is controlled by this variable There are 16 different types of testbenches that can be generated.

The allowable testbench types are:

symbolic	Generates a fully symbolic testbench.
binary	Generates a binary testbench. No symbolic or flush cycles.

cam	Generates a suite of testbenches suite for content addressable memories. Includes cammatch testbench.
cammatch	CAM match integrity test
clkenum	Clock enumeration
clkwave	Clock wave
dataint	Data integrity
dualphs	Two phase - assert twice, check twice
holdchk	Hold time X value test
library	Generates a separate testbench targeted for each cell in a cell library. See the manual page **library_verification** for more information on using this style of testbench.
macro	Generates a suite of testbenches suited for a combinatorial design
portcov	Multiple data integrity tests for each clock
protocol	Protocol
rom	Generates a set of testbenches targeted for ROM type designs
romhold	Dual phase ROM hold test
romproto	Rom protocol test
sram	Generates a set of testbenches targeted for SRAM type designs
mapping	Generates a set of testbenches targeted for a logical to SPICE netlist mapping application

The generated testbench will have a variable `esp_testbench_style` that is set to a specific value based upon the testbench style. The *cam*, *macro*, *rom* and *sram* style generate more than one testbench. The following table shows the specific values set for `esp_testbench_style` and the default suffix of the testbench file.

Style	**Suffix**	**esp_testbench_style**
symbolic	.sym	symbolic
binary	.bin	binary
cammatch	.mat	cammatch
clkenum	.clk	clkenum
clkwave	.mpt	clkwave
dataint	.dit	dataint
dualphs	.2ph	dualphs
holdchk	.hld	holdchk
library	.ltb	library
portcov	.dit_N_M where N and M are positive integers	portcov
protocol	.ptl	protocol
romhold	.rom	romhold
romproto	.ptl	romproto

The **set_constraint** command option *-style* checks the `esp_testbench_style` variable in the testbench.

Number of Cycles per Testbench Type

The Tcl testbench styles macro, sram, rom, and cam generate a suite of test benches using hard coded numbers of binary, symbolic and flush cycles. The value of Tcl variables for testbench cycles are completely ignored for these test benches.

Testbench style	type	Number of cycles per testbench binary	symbolic	flush
sram	bin	8	0	2
	dit	8	3	2
	ptl	8	4	2
	2ph	8	4	2
rom	bin	8	0	1
	ptl	8	1	1
	2ph	8	1	1
cam	bin	8	0	2
	dit	8	3	2
	ptl	8	6	2
	2ph	8	6	2
	mat	8	3	2
macro	bin	8	0	3
	ptl	8	1	3
	2ph	8	1	3

The Tcl testbench styles symbolic, dataint, protocol, dualphs, clkenum, clkwave, romhold, cammatch, holdchk, romproto, portcov generate a single testbench. These single testbenches can be controlled by the testbench cycle variables.

- **b** represents value of **testbench_binary_cycles**
- **s** represents value of **testbench_symbolic_cycles**
- **f** represents value of **testbench_flush_cycles**

Testbench style	type	Number of cycles per testbench binary	symbolic	flush
binary	bin	b	s	f
cammatch[1]	mat	b	3	f
clkenum	clk	b	s	f
clkwave	mpt	b	s	f
dataint[1]	dit	b	3	f
dualphs	2ph	b	s	f
holdchk	hld	b	s	f
library[3]	ltb	b	s	f
portcov[1]	dit	b	3	f
protocol	ptl	b	s	f

romhold[2]	rom	b	1	1
romproto[2]	rom	b	1	1
symbolic	sym	b	s	f

Note:

1. cammatch, dataint and portcov are fixed at 3 symbolic cycles.
2. romhold and romproto are fixed at 1 symbolic and 1 flush cycle
3. library - typically only 2 binary and 0 flush cycles needed. Symbolic cycles can be as few as 2 for pure combinatorial cells. Complex multiple bit cells need more than 5 symbolic cycles.

SEE ALSO

Commands:
write_testbench
set_constraint

Variables:
testbench_binary_cycles
testbench_constraint_file
testbench_declaration_file
testbench_dump_symbolic_waveform
testbench_flush_cycles
testbench_implementation_instance
testbench_initialization_file
testbench_module_name
testbench_output_checks
testbench_reference_instance
testbench_symbolic_cycles

testbench_symbolic_cycles

NAME

testbench_symbolic_cycles

Number of symbolic cycles used in standard testbench.

TYPE

integer

Legal values: >=0

DEFAULT

3

DESCRIPTION

Defines the number of symbolic cycles in the testbench created by **write_testbench** .

If more than 10 symbolic cycles are needed, a custom testbench should be created instead of using the testbench created by **write_testbench** .

Note:

testbench_style is affected by this variable. Testbench styles of macro, sram, rom and cam ignore the setting of this variable.

SEE ALSO

Commands:
write_testbench

Variables:
testbench_binary_cycles
testbench_constraint_file
testbench_declaration_file
testbench_dump_symbolic_waveform
testbench_flush_cycles
testbench_implementation_instance
testbench_initialization_file
testbench_module_name
testbench_output_checks
testbench_reference_instance
testbench_style

threshold_high_impedance_resistance

NAME

threshold_high_impedance_resistance

Resistance threshold for a floating net

TYPE

integer

Legal values: 0 to 2,147,483,647

DEFAULT

0

DESCRIPTION

When **threshold_high_impedance_resistance** is 0 no driving paths are ignored.

When **threshold_high_impedance_resistance** is larger than 0, the value specifies that driving paths with resistances larger than the value are to be ignored. If no driving path has a resistance less than **threshold_high_impedance_resistance** then the net will have a Verilog Z value (also known as floating).

This is useful for validating SPICE models with weak pull-up or pull-down drivers to produce Verilog Z values.

This variable affects RC mode analysis.

SEE ALSO

Commands: **read_spice** **verify**

threshold_warn_big_rc_delay

NAME

threshold_warn_big_rc_delay

Issue warnings when delays are longer than this.

TYPE

integer

Legal values: 0 to 2,147,483,647

DEFAULT

0

DESCRIPTION

Delays longer than **threshold_warn_big_rc_delay** picoseconds will cause a warning message to be issued.

The default value of 0 means no warning is issued.

SEE ALSO

Commands: **verify**

verdi_path

NAME

verdi_path

Command to start the verdi viewer

TYPE

string

Legal value: Bourne shell script

DEFAULT

verdi

DESCRIPTION

Specifies a command (or simple script) to start the Verdi waveform viewer for debugging.

This variable is used by **explore_with_viewer** to start a graphical interactive signal trace debug session.

Example

Verdi is not in the path. Verdi is executed from /usr/tools/synopsys/verdi/bin/verdi.

```
> set_app_var verdi_path /usr/tools/synopsys/verdi/bin/verdi
/usr/tools/synopsys/verdi/bin/verdi
```

SEE ALSO

Commands: **debug_design** **explore_with_viewer** **start_waveform_viewer** **verify**

Variables: **waveform_format**

Topics: **dve** **fsdb** **vcd**

verify_auto_effort_selection

NAME

verify_auto_effort_selection

Attempt lower-effort simulation models for faster verification. If error found then automatically raise effort level.

TYPE

Boolean

Legal values: *true false*

DEFAULT

false

DESCRIPTION

When *true* the ESP tool automatically selects the lowest effort level possible to show equivalence between the Verilog and SPICE designs, possibly resulting in faster verification. If the lower effort levels result in error vectors, the effort level is increased. Because the selection process has some overhead, designs that normally take longer than 30 minutes see the largest benefit. In some cases, the ESP tool determines that only the highest effort level is sufficient and the runtime may increase by about 10 percent. Binary debug simulation runs (**debug_design** always run in the high effort mode, because symbolic runs that generate counter-examples always terminate at the highest effort level.

If **verify_max_number_error_vectors** has been set to a value greater than the default of 1 then

verify_auto_effort_selection cannot be set to *true*. If verify_auto_effort_selection has already been set to *true* and then **verify_max_number_error_vectors** is set to something greater than 1 then **verify_auto_effort_selection** will be set to *false* and a warning message to that effect will be issued.

SEE ALSO

Commands: **verify**

Variables: **verify_delay_round_multiple** **verify_dynamic_reorder** **verify_max_number_error_vectors** **verify_r**

verify_coverage_max_dropped_symbol_levels

NAME

verify_coverage_max_dropped_symbol_levels

Sets the number of levels for dropped symbol coverage recording.

TYPE

integer

Legal values: 0 to 2,147,483,647

DEFAULT

1

DESCRIPTION

This variable specifies the number of levels of hierarchy for which dropped symbol coverage data will be recorded in the database.

The default value of 1 records dropped symbols coverage data only for the top level of the design.

A value of 0 specifies no limit. All levels of the design hierarchy will have dropped symbol coverage data recorded in the coverage database. However, recording dropped symbol coverage data for all levels of hierarchy can be expensive in terms of simulation time and memory.

In order to see the coverage results for lower levels of the design you will need to use the "select" feature

of the coverage report. See the flow man page **coverage_reports** for information on setting the scope for dropped symbol reporting. You will need to have a coverage spec file that looks like:

```
printdetail NS SA LC DS
select (0,inno_tb_top.xref)
select (0,inno_tb_top.ximp)
select (0,inno_tb_top.ximp.X1)
select (0,inno_tb_top.ximp.X2)
select (0,inno_tb_top.ximp.X3)
  ...
```

SEE ALSO

Commands: **report_coverage**

Variables: **coverage**

Topic: **coverage_filters** **coverage_merge** **coverage_overview** **coverage_reports** **coverage_tasks** **coverage_usag**

verify_delay_round_multiple

NAME

verify_delay_round_multiple

RC delays are rounded to this value.

TYPE

integer

Legal values: 1 to 2,147,483,647

DEFAULT

1. picoseconds

DESCRIPTION

Setting **verify_delay_round_multiple** controls the rounding of RC delays. RC calculated delays are rounded to the nearest **verify_delay_round_multiple** picoseconds.

The **verify_rcdelay_unit** variable controls if the rounding is in picoseconds or femtoseconds.

Setting this number to less than the default will provide more accurate delays at the expense of simulation time.

Setting this number to more than the default will speed up simulation at the expense of timing accuracy.

Values of 50 or even 100 can be used to increase simulation speed. Caution should be exercised in tuning simulation performance using this variable as timing accuracy can affect functionality of many designs.

For technology nodes under 40nm you most likely want to set this value to 1ps.

EXAMPLE

Set delay rounding to 1 picosecond. The default value for **verify_rcdelay_unit** is picoseconds.

```
> set_app_var verify_delay_round_multiple 1
1
```

Set delay rounding to 100 femtoseconds for a 14nm process.

```
> set_app_var verify_rcdelay_unit fs
fs
> set_app_var verify_delay_round_multiple 100
100
```

SEE ALSO

Commands: **read_spice** **report_design_spice_mode** **reset_process** **set_design_spice_mode** **set_instance_delay** **s**

Variables: **netlist_inverter_chain_extraction** **verify_rcdelay_unit**

verify_dynamic_reorder

NAME

verify_dynamic_reorder

Controls dynamic reordering of symbols

TYPE

string

Legal values: *on off conservative auto*

DEFAULT

auto

DESCRIPTION

The order of evaluation of symbols when creating equations affects the storage size for the equation. The variable **verify_dynamic_reorder** controls the dynamic reordering of symbol evaluation.

If **verify_dynamic_reorder** is *auto* reordering starts as *off* and automatically switches to conservative and restarts simulation if randomization is about to occur. However, if RC mode is being used and **verify_hierarchical_compression** is not *off* then *conservative* is used and restart is disabled.

Randomization is controlled by the **verify_randomize_variables** variable.

SEE ALSO

Commands: **verify**

Variables: **verify_hierarchical_compression** **verify_randomize_variables**

verify_feedback_detection

NAME

verify_feedback_detection

Enable feedback net detection.

TYPE

Boolean

Legal values: *true false*

DEFAULT

true

DESCRIPTION

ESP has a algorithm that attempts to detect nets with feedback. ESP will reorder events on such nets to avoid a race condition that could cause the net to float or have an *X* value.

This reordering of events happens only for simultaneous events and is permitted by the IEEE SystemVerilog Language Reference Manual.

Generally, **verify_feedback_detection** should be *true*.

SEE ALSO

Commands: **verify**

verify_glitch_removal

NAME

verify_glitch_removal

Controls glitch suppression.

TYPE

string

Legal values: *on off aggressive*

DEFAULT

on

DESCRIPTION

Controls glitch suppression.

The default value of *on* conforms to the IEEE Verilog Language Reference Manual's description for glitch removal. Input pulses that would cause an output change where the pulse width is less than the path delay will be suppressed.

The value *off* specifies that output glitches on modules with specify blocks will not be suppressed.

The value *aggressive* causes outputs glitches to be suppressed when the input pulse is less than or equal to the path delay.

It is believed that Cadence NC-Verilog uses *aggressive* glitch suppression and not ESP's default *on* glitch suppression.

The Verilog LRM uses the term pulse suppression. Section 14.6 of IEEE 1364-2001 discusses pulse filtering and the inertial delay model.

SEE ALSO

Commands: **verify**

verify_hierarchical_compression

NAME

verify_hierarchical_compression

Controls the level of hierarchical compression.

TYPE

string

Legal values: *off low medium high ACT0*

DEFAULT

high

DESCRIPTION

Hierarchical compression is controlled by the **verify_hierarchical_compression** variable.

Note:
In general, do not change the compression level. The default value is sufficient for most designs.Talk to Synopsys support if you think you have to change this variable.

Using hierarchical compression allows ESP to leverage any repetitive structures present in the design. Synopsys has developed a method of storing these repetitive structures in a compact manner that greatly reduces the physical memory required for simulation and decreases the CPU time required.

The **verify_hierarchical_compression** variable controls the number of nets or primitives that must be present before the hierarchy is considered worthy of compression.

The value to use is design type dependent, but experience has shown that the *high* value works best for SRAM-type designs that have hierarchical cores.

If the core of the SRAM is flat, *ACT0* is appropriate.

With compression there is a risk. If there are too many duplicated instances and those instances are small blocks, the overhead of compression itself might take more memory than that required by uncompressed blocks.

A value of *off* turns off hierarchical compression.

A value of *low* specifies that the design must have large blocks that are reused before considering compression.

A value of *medium* considers medium-size blocks for compression.

A value of *high* allows smaller blocks being reused to be compressed, but does not compress blocks with a few transistors. Examples of blocks with a few transistors are simple memory cells and basic logic gates, such as inverters or nands. This is the best setting for SRAM-type circuits with hierarchical cores.

A value of *ACT0* (the last character is the number zero) allows any hierarchy to be compressed if it is reused.

SEE ALSO

Commands: **verify**

verify_imitate_simulator

NAME

verify_imitate_simulator

Tells ESP to behave closer to a specific simulator

TYPE

string

Legal values: *esp*, *vcs*, *nc*, *inno*

DEFAULT

esp

DESCRIPTION

Tells ESP to simulate using the semantics for a specific Verilog simulator. By default, the ESP interpretation of the Verilog Language Reference is optimized for fast efficient symbolic simulation.

The Verilog Language Reference has numerous ambiguities that result in differences in evaluation ordering and other semantic interpretations in conforming simulators.

Using **verify_imitate_simulator** the user can cause ESP to match closely as possible to the chosen simulator.

None of the imitations are perfect. Race conditions are possible that can not be matched. The solution to

such differences is to recode the Verilog source to eliminate the race conditions.

If you are using VCS as your Verilog simulator and you want ESP to match as closely as possible to VCS then set **verify_imitate_simulator** to *vcs*.

If you are using NC-Verilog as your Verilog simulator and you want ESP to match as closely as possible to NC-Verilog then set **verify_imitate_simulator** to *nc*.

If you want ESP to match the behavior in the A-2007.12-SP1 release then set **verify_imitate_simulator** to *inno*.

SEE ALSO

Commands: **verify**

verify_max_number_error_vectors

NAME

verify_max_number_error_vectors

Maximum number of error vectors that will be reported per testbench.

TYPE

Integer

Legal values: >=0

DEFAULT

1

DESCRIPTION

Defines the maximum number of error vectors that will be reported per testbench when the **verify** command finishes. If the value is greater than 1 then ESP will try to find multiple error vectors that illustrate distinct differences.

SEE ALSO

Commands: **debug_design** **verify**

verify_max_number_of_oscillations

NAME

verify_max_number_of_oscillations

Sets the limit on net oscillations.

TYPE

integer

Legal values: >=0

DEFAULT

1.

DESCRIPTION

Sets the limit on the number of times a net can change values in a single simulation time step.

When **verify_max_number_of_oscillations** is exceeded, the simulator gives up and declares an oscillation has been detected.

This value should not have to be changed as the default is an arbitrarily large number.

SEE ALSO

Commands: **verify**

Variables: **verify_max_number_oscillation_vectors** , **verify_max_reported_oscillations** , **verify_stop_on_n**

verify_max_number_oscillation_vectors

NAME

verify_max_number_oscillation_vectors

Maximum number of oscillation vectors that will be reported per testbench.

TYPE

Integer

Legal values: >= 0

DEFAULT

1

DESCRIPTION

Defines the maximum number of oscillation vectors that will be reported per testbench when the **verify** command finishes. If the value is greater than 1 then ESP will try to find multiple oscillation vectors that illustrate distinct oscillation paths.

The **verify_max_number_of_oscillations** variable controls the threshold for generation of an oscillation vector.

The **verify_max_reported_oscillations** variable controls the number of oscillation events that are reported in the log file.

An oscillation event is a net that changes values more than **verify_max_number_of_oscillations** times within a single simulation time step.

SEE ALSO

Commands: **debug_design** **report_error_vectors** **verify**

Variables: **verify_auto_effort_selection** **verify_max_number_of_oscillations** **verify_max_reported_oscillat**

verify_max_reported_oscillations

NAME

verify_max_reported_oscillations

Controls the maximum number of oscillations reported.

TYPE

integer

Legal values: >=0

DEFAULT

4

DESCRIPTION

This controls the number of oscillating nets that will be reported.

SEE ALSO

Commands: **verify**

Variables: verify_max_number_of_oscillations verify_max_number_oscillation_vectors verify_stop_on_nonz

verify_negative_timing_checks

NAME

verify_negative_timing_checks

Enable the use of negative timing check values

TYPE

Boolean

Legal values: *on off*

DEFAULT

off

DESCRIPTION

Negative timing values can be in the Verilog source files or in the SDF annotation file.

By default negative values in timing checks are interpreted as 0.

Setting **verify_negative_timing_checks** to *on* enables the actual use of negative values in timing checks such as `$setuphold()` and `$recrem()`.

SEE ALSO

Commands: **verify**

verify_partial_transition_sensitivity

NAME

verify_partial_transition_sensitivity

Controls sensistivity factor for transitioning nets

TYPE

integer

Legal values: >=0

DEFAULT

1. indicating use of internally computed default value.

DESCRIPTION

Sensitivity of transistor drive strength to transitioning gate signals. Suggested values are between 2 and 5.

SEE ALSO

Variables: **verify_partial_transition_threshold**

verify_partial_transition_threshold

NAME

verify_partial_transition_threshold

Specifies the delay time that triggers a net transitioning status

TYPE

integer

Legal values: >=0

DEFAULT

1. Indicating use of internally calculated threshold.

DESCRIPTION

Delay threshold (in ps) for modeling partial transitions in the implementation model. Suggested value is approximately 3 inverter delays.

Delays longer than this value will be flagged as transitioning.

SEE ALSO

Variables: **verify_partial_transition_sensitivity**

verify_randomize_variables

NAME

verify_randomize_variables

Controls amount of memory allocated for symbols

TYPE

string

Legal values: *on off low medium high sysmem*

DEFAULT

on

DESCRIPTION

Switches the possibility for randomizing of variables on or off and controls the tolerance for randomization.

Randomization will occur when a threshold for the amount of memory used by the ESP tool for holding symbolic equations is reached. During randomization the ESP tool randomly selects a symbolic variable and randomly sets it to a binary value of *0* or *1* . A message is printed stating which symbolic value has been randomized and to what value it has been set. The file *esp.Random* is created and contains a list of all randomized symbols and their respective values.

Generally, it is not a good idea to let a verification randomize. Randomization represents a reduction in the functional coverage of the testbench. Divide-and-Conquer techniques should be used to eliminate

randomization, though you may want to try turning on Auto Effort Selection using the **verify_auto_effort_selection** variable.

The value *high* allows randomization to occur more easily (you have a high tolerance for randomization--you are very willing to let randomization happen) resulting in less run time but more variables randomized.

The value *medium* tries to avoid randomization, but randomization is allowed.

The value *low* tries very hard to avoid randomization (you have a low tolerance for it) but it can happen if the simulation gets too complex. This results in more run time but fewer randomized variables.

The value *on* is the default level. It is a value between *medium* and *low* .

The value *sysmem* uses the available physical memory to avoid randomization as much as possible, randomizing only when almost all the memory of the system is consumed by the ESP tool.

The value *off* forbids randomization entirely. With this level, you run the risk of using too much memory and CPU time. Using more memory than is available causes an out-of-memory termination. Use *off* only as a last resort.

If you think you need *off* consider instead restructuring your verification to use fewer symbols. Use and divide and conquer approach with more testbenches.

SEE ALSO

Commands: **verify**

Variables: **verify_stop_on_randomize** **verify_auto_effort_selection**

verify_rcdelay_unit

NAME

verify_rcdelay_unit

The RC time unit

TYPE

String

Legal values: *ps p f fs*

DEFAULT

ps

DESCRIPTION

This variable controls the units of SPICE delay times. The default is pico seconds. Use this variable with **verify_delay_round_multiple** to control rounding of delay values in the SPICE netlist.

EXAMPLES

To get delay values rounded to 10 ps use:

```
set_app_var verify_delay_round_multiple 10
set_app_var verify_rcdelay_unit ps
```

Note:
This is the actual default, so you do not have to use these commands

For a design under 40nm, set the delay rounding to 1 ps with:

```
set_app_var verify_delay_round_multiple 1
set_app_var verify_rcdelay_unit ps
```

For a design under 14nm, set delay rounding to 100 fs with:

```
set_app_var verify_delay_round_multiple 100
set_app_var verify_rcdelay_unit fs
```

SEE ALSO

Commands: **read_spice** **report_design_spice_mode** **reset_process** **set_design_spice_mode** **set_instance_delay** **s**

Variables: **verify_delay_round_multiple**

Other: **device_model_simulation**

verify_spice_simulation_mode

NAME

verify_spice_simulation_mode

Globally set the SPICE simulation mode.

TYPE

String

Legal values are *rcdelays*, *rcunit*, *rcstrength*, *rcoff*, *switch*, *pwl_1*

DEFAULT

rcdelays

DESCRIPTION

Sets how ESP interprets the full SPICE netlist.

Generally use the default value. The other modes (except *pwl_l*) can provide faster verifications at the cost of more scripting to properly configure a SPICE netlist to produce a functionally correct verification.

Mode	
rcdelays	device sizes affect delays and drive fights
rcunit	device sizes affect drive fights but delays are 1 ps
rcstrength	device sizes affect drive fights but delays are 0 ps

rcoff	Verilog switch
switch	Switch model
pwl_1	Internal use

The *switch* mode and *rcoff* modes are very similar. In *switch* mode, ESP maintains the device length and width information within the internal SPICE netlist. The *rcoff* mode is a pure Verilog switch model that has no transistor length and width annotations.

The **set_design_spice_mode** and **set_instance_spice_mode** commands are used to selectively control the interpetation of SPICE sub-circuits and sub-circuit instantiations.

SEE ALSO

Commands" **read_spice** **set_design_spice_mode** **set_instance_spice_mode**

verify_stop_on_nonzero_delay_oscillation

NAME

verify_stop_on_nonzero_delay_oscillation

Stop simulation if non-zero delay oscillations occur.

TYPE

Boolean

Legal values: *true false*

DEFAULT

false

DESCRIPTION

If *true* , simulation will stop if more than 1024 non-zero delay oscillations occur on the SPICE parts of the design.

SEE ALSO

Commands: **verify**

Variables: **verify_max_number_of_oscillations** **verify_max_number_oscillation_vectors** **verify_max_reported**

verify_stop_on_randomize

NAME

verify_stop_on_randomize

Stop simulation if randomization occurs.

TYPE

Boolean

Legal values: *true false*

DEFAULT

true

DESCRIPTION

If *true*, simulation will stop if randomization occurs.

SEE ALSO

Commands: **verify**

Variables: **verify_randomize_variables**

verify_stop_on_zero_delay_oscillation

NAME

verify_stop_on_nonzero_delay_oscillation

Stop simulation if zero delay oscillations occur.

TYPE

Boolean

Legal values: *true false*

DEFAULT

false

DESCRIPTION

If *true* simulation will stop if zero delay oscillations occur.

The **verify_max_number_of_oscillations** variable controls the number of oscillations that have to happen before the simulator will stop.

SEE ALSO

Commands: **verify**

Variables: **verify_max_number_of_oscillations** **verify_max_number_oscillation_vectors** **verify_max_reported**

verify_suppress_nonzero_delay_oscillation

NAME

verify_suppress_nonzero_delay_oscillation

Suppresses the non-zero delay oscillations in the implementation model.

TYPE

Boolean

Valid values: *true false*

DEFAULT

true

DESCRIPTION

When you set the **verify_suppress_nonzero_delay_oscillation** variable to *true* , the oscillations of an RC mode node that is involved in a non-zero delay oscillation for more than 1024 transitions is suppressed by ESP by forcing the node to the value X. To change this behavior and allow the oscillations of this type to continue indefinitely, set the variable value to *false* .

SEE ALSO

Commands: **verify**

Variables: **verify_max_number_of_oscillations** **verify_max_number_oscillation_vectors** **verify_max_reported**

verify_use_specify

NAME

verify_use_specify

Use Verilog specify blocks during verify.

TYPE

Boolean

Legal values: *true false*

DEFAULT

true

DESCRIPTION

If *true* Verilog specify blocks will be simulated.

If *false* Verilog specify blocks will not be simulated. This is the same as the VCS `+nospecify` switch.

SEE ALSO

Commands: **verify**

waveform_dump_control

NAME

waveform_dump_control

Controls the automatic dumping of waveforms during verification and debug.

TYPE

string

Legal values: *on off*

DEFAULT

on if in verify mode *compare* or *inspector* and *off* if in verify mode *simulate* .

DESCRIPTION

If the variable value is *on* , then automatic dumping of waveforms is controlled by the Tcl shell. If the variable value is *off* , then the user must control the dumping of waveform data.

The **set_verify_mode** command will change the value of this variable and the user is responsible for setting the value to the desired setting after the verification mode has been changed.

`set_verify_mode compare` will set the value of **waveform_dump_control** to *on* . `set_verify_mode inspector` will set the value of **waveform_dump_control** to *on* . `set_verify_mode simulate` will set the value of **waveform_dump_control** to *off* .

SEE ALSO

Commands: **set_verify_mode** **debug_design** **verify**

Variables: **testbench_implementation_instance** **testbench_module_name** **testbench_reference_instance**

waveform_format

NAME

waveform_format

Identify the waveform data dump type format.

TYPE

string

Legal values: *vcd fsdb*

DEFAULT

vcd

DESCRIPTION

Defines the waveform dump file format that will be used for binary debug simulations.

This option controls the dump commands that are are written to the default scope file.

vcd is the Verilog value change dump format. This format is supported by all Verilog simulators.

fsdb is a compressed file format supported by Verdi. ESP can dump signal values in this format. This format supports dumping of real values and signal strengths.

vpd is a compressed file format supported by Synopsys VCS, Magellan and DVE. ESP does not support

dumping to *vpd* format in this release.

The "scope file" is a special Verilog module that defines the name of the **vcd** file and the nets that are to be dumped. The default "scope file" is:

```
module espscopedef();
  initial begin
    $dumpfile("dump.vcd");
    $dumpvars(0, inno_tb_top);
  end
endmodule
```

To change the name of the created **vcd** file or to change the nets dumped to the **vcd** file, create a scope file and use the *-dumpscope* switch of **debug_design** to select the created file.

Example

Use the FSDB format with Verdi.

```
> set_app_var waveform_format fsdb
fsdb
> set_app_var waveform_viewer {verdi %s -ssy -ssv -ssz -ssf %v}
verdi %s -ssy -ssv -ssz -ssf %v
```

SEE ALSO

Commands: **debug_design** **start_waveform_viewer**

Variables: **waveform_viewer**

Topics: **vcd**

waveform_viewer

NAME

waveform_viewer

Command to start the waveform viewer

TYPE

string

Legal value: Bourne shell script where *%s* is replaced by a list of Verilog files and *%v* is replaced by the base name of the waveform dump file.

DEFAULT

vfast %v; verdi %s -ssy -ssv -ssz -ssf %v.fsdb

DESCRIPTION

Specifies a command (or simple script) to start a waveform viewer for debugging.

The value *%s* if present will be replaced by the list of Verilog files.

The value *%v* if present will be replaced by the basename of the waveform dump file.

The default value invokes Verdi.

This variable is used by **start_waveform_viewer** to start a waveform debugging session.

Example

Use the FSDB format with Verdi.

```
> set_app_var waveform_format fsdb
fsdb
> set_app_var waveform_viewer {verdi %s -ssy -ssv -ssz -ssf %v}
verdi %s -ssy -ssv -ssz -ssf %v
```

SEE ALSO

Commands: **debug_design** **start_waveform_viewer** **verify**

Variables: **waveform_format**

Topics: **dve** **fsdb** **vcd**