



ModelSim® SE Reference Manual

Software Version 6.6a

**© 1991-2010 Mentor Graphics Corporation
All rights reserved.**

This document contains information that is proprietary to Mentor Graphics Corporation. The original recipient of this document may duplicate this document in whole or in part for internal business purposes only, provided that this entire notice appears in all copies. In duplicating any part of this document, the recipient agrees to make every reasonable effort to prevent the unauthorized use and distribution of the proprietary information.

This document is for information and instruction purposes. Mentor Graphics reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Mentor Graphics to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Mentor Graphics products are set forth in written agreements between Mentor Graphics and its customers. No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Mentor Graphics whatsoever.

MENTOR GRAPHICS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

MENTOR GRAPHICS SHALL NOT BE LIABLE FOR ANY INCIDENTAL, INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES WHATSOEVER (INCLUDING BUT NOT LIMITED TO LOST PROFITS) ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF MENTOR GRAPHICS CORPORATION HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

RESTRICTED RIGHTS LEGEND 03/97

U.S. Government Restricted Rights. The SOFTWARE and documentation have been developed entirely at private expense and are commercial computer software provided with restricted rights. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement provided with the software pursuant to DFARS 227.7202-3(a) or as set forth in subparagraph (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable.

Contractor/manufacturer is:

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777.
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

Send Feedback on Documentation: supportnet.mentor.com/user/feedback_form.cfm

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Mentor Graphics Corporation or other third parties. No one is permitted to use these Marks without the prior written consent of Mentor Graphics or the respective third-party owner. The use herein of a third-party Mark is not an attempt to indicate Mentor Graphics as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A current list of Mentor Graphics' trademarks may be viewed at: www.mentor.com/terms_conditions/trademarks.cfm.

Table of Contents

Chapter 1

Syntax and Conventions	15
Documentation Conventions	15
File and Directory Pathnames	16
Design Object Names	16
Object Name Syntax	16
SystemC Class, Structure, and Union Member Specification	17
SystemVerilog Scope Resolution Operator	18
Specifying Names	19
Escaping Brackets and Spaces in Array Slices	21
Environment Variables and Pathnames	21
Name Case Sensitivity	21
Extended Identifiers	22
Wildcard Characters	22
Using the WildcardFilter Preference Variable	23
Simulator Variables	28
Simulation Time Units	28
Argument Files	29
Command Shortcuts	30
Command History Shortcuts	30
Numbering Conventions	31
VHDL Numbering Conventions	31
Verilog Numbering Conventions	32
GUI_expression_format	32
Expression Typing	33
Expression Syntax	33
Signal and Subelement Naming Conventions	39
Grouping and Precedence	39
Concatenation of Signals or Subelements	39
Record Field Members	41
Searching for Binary Signal Values in the GUI	41

Chapter 2

Commands	43
.main clear	57
abort	58
add atv	59
add button	60
add dataflow	63
add list	65
add memory	70
add schematic	72

add testbrowser	74
add watch	75
add wave	76
add_cmdhelp	83
add_menu	85
add_menucb	87
add_menuitem	89
add_separator	91
add_submenu	92
alias	94
assertion action	95
assertion active	97
assertion count	99
assertion enable	101
assertion fail	104
assertion pass	106
assertion profile	108
atv log	110
batch_mode	112
bd	113
bookmark add wave	115
bookmark delete wave	117
bookmark goto wave	118
bookmark list wave	119
bp	120
cd	127
cdbg	128
change	131
change_menu_cmd	134
check contention add	135
check contention config	137
check contention off	138
check float add	139
check float config	141
check float off	142
check stable off	143
check stable on	144
checkpoint	146
compare add	148
compare annotate	154
compare clock	156
compare configure	158
compare continue	160
compare delete	161
compare end	162
compare info	163
compare list	165
compare options	166
compare reload	171

Table of Contents

compare reset	172
compare run	173
compare savediffs	174
compare saverules	175
compare see	176
compare start	178
compare stop	180
compare update	181
configure	182
context	188
coverage analyze	191
coverage attribute	198
coverage clear	201
coverage edit	203
coverage exclude	205
coverage goal	212
coverage open	215
coverage ranktest	216
coverage report	221
coverage save	232
coverage tag	234
coverage testnames	238
coverage unlinked	239
coverage weight	242
dataset alias	245
dataset clear	246
dataset close	247
dataset config	248
dataset current	250
dataset info	251
dataset list	252
dataset open	253
dataset rename	254
dataset restart	255
dataset save	256
dataset snapshot	257
delete	260
describe	261
disablebp	262
disable_menu	263
disable_menuitem	264
do	265
down	267
drivers	270
dumplog64	271
echo	272
edit	273
enablebp	274
enable_menu	275

enable_menuitem	276
encoding	277
environment	278
examine	280
exit	285
fcover configure	286
find	289
find drivers	294
find infiles	299
find insource	300
formatTime	302
force	303
fsm list	307
fsm properties	308
fsm view	309
gdb dir	310
getactivecursortime	311
getactivemarkertime	312
help	313
history	314
jobspy	315
layout	316
lecho	318
left	319
log	322
lshift	326
lsublist	327
mem compare	328
mem display	329
mem list	332
mem load	333
mem save	337
mem search	340
messages clearfilter	343
messages setfilter	344
modelsim	345
next	346
noforce	347
nolog	348
notepad	350
noview	351
nowhen	352
onbreak	353
onElabError	355
onerror	356
onfinish	358
pa msg	359
pause	361
pop	362

Table of Contents

power add	363
power off	365
power on	367
power report	369
power reset	372
precision	374
printenv	375
process report	376
profile clear	377
profile interval	378
profile off	379
profile on	380
profile open	382
profile option	383
profile reload	385
profile report	386
profile save	390
project	391
property list	394
property wave	396
push	398
pwd	399
quietly	400
quit	401
qverilog	402
radix	404
radix define	406
radix delete	409
radix list	410
radix names	411
radix signal	412
readers	413
report	414
restart	416
restore	418
resume	419
right	420
run	423
runStatus	426
sccom	428
scgenmod	436
sdfcom	439
search	441
searchlog	444
see	447
seetime	448
setenv	450
shift	451
show	452

simstats	453
status	455
step	456
stop	457
suppress	458
tb	459
tcheck_set	460
tcheck_status	464
Time	467
toggle add	470
toggle disable	473
toggle enable	474
toggle report	475
toggle reset	477
tr color	478
tr order	481
tr uid	483
transcribe	485
transcript	486
transcript file	487
transcript path	489
transcript sizelimit	490
tssi2mti	491
typespec	492
ui_VVMode	493
unsetenv	495
up	496
vcd add	498
vcd checkpoint	500
vcd comment	501
vcd dumpports	502
vcd dumpportsall	505
vcd dumpportsflush	506
vcd dumpportslimit	507
vcd dumpportsoff	508
vcd dumpportson	509
vcd file	510
vcd files	512
vcd flush	514
vcd limit	515
vcd off	516
vcd on	517
vcd2wlf	518
vcom	520
vcover attribute	538
vcover dump	540
vcover merge	542
vcover ranktest	548
vcover report	554

Table of Contents

vcover stats	565
vcover testnames	568
vdbg	570
vdel	571
vdir	573
vencrypt	576
verror	578
vgencomp	580
vhencrypt	582
view	584
virtual count	588
virtual define	589
virtual delete	590
virtual describe	591
virtual expand	592
virtual function	593
virtual hide	596
virtual log	597
virtual nohide	599
virtual nolog	600
virtual region	602
virtual save	603
virtual show	604
virtual signal	605
virtual type	609
vlib	611
vlog	613
vmake	639
vmap	641
vopt	642
vsim	676
vsim<info>	715
vsim_break	716
vsources	717
wave	718
wave create	722
wave edit	728
wave export	731
wave import	732
wave modify	733
wave sort	736
when	737
where	744
wlf2log	745
wlf2vcd	748
wlfman	749
wlfrecover	753
write cell_report	754
write format	755

write list	757
write preferences	758
write report	759
write timing	762
write transcript	763
write tssi	764
write wave	766
xml2ucdb	768
Chapter 3	
AVM Encyclopedia	773
Class Index	773
Classes for Components	778
avm_env	779
avm_named_component	781
avm_random_stimulus	786
avm_stimulus	788
avm_subscriber	789
avm_threaded_component	790
avm_verification_component	791
Classes for Comparators	791
avm_algorithmic_comparator	793
avm_in_order_built_in_comparator	795
avm_in_order_class_comparator	796
avm_in_order_comparator	797
Classes for Connectors	799
avm*_export	800
avm*_imp	802
avm*_port	804
avm_analysis_port	806
avm_blocking_master_imp	807
avm_blocking_slave_imp	809
avm_connector_base	811
avm_master_imp	815
avm_nonblocking_master_imp	817
avm_nonblocking_slave_imp	819
avm_port_base	821
avm_slave_imp	823
avm_transport_imp	825
analysis_imp	826
analysis_port	827
global_analysis_ports	828
tlm*_imp	829
Classes for Channels	829
analysis_fifo	831
tlm_fifo	832
tlm_req_rsp_channel	835
tlm_transport_channel	838

Table of Contents

TLM Interfaces	838
analysis_if #(type T=int)	839
tlm_blocking_get_if	840
tlm_blocking_get_peek_if	841
tlm_blocking_peek_if	842
tlm_blocking_put_if	843
tlm_blocking_master_if	844
tlm_blocking_slave_if	845
tlm_get_if	846
tlm_get_peek_if	847
tlm_master_if	849
tlm_nonblocking_get_if	851
tlm_nonblocking_get_peek_if	852
tlm_nonblocking_master_if	853
tlm_nonblocking_peek_if	855
tlm_nonblocking_put_if	856
tlm_nonblocking_slave_if	857
tlm_peek_if	859
tlm_put_if	860
tlm_slave_if	861
tlm_transport_if	863
Transactions	863
avm_built_in_clone	864
avm_built_in_comp	865
avm_built_in_converter	866
avm_built_in_pair	867
avm_class_clone	868
avm_class_comp	869
avm_class_converter	870
avm_class_pair	871
avm_transaction	872
Reporting	873
avm_report_client	874
avm_report_handler	879
avm_report_server	882
avm_reporter	884

Index

End-User License Agreement

List of Examples

Example 1-1. Base- and Descendant-Class Specification 17
Example 1-2. SystemVerilog Scope Resolution Operator Example 19

List of Figures

Figure 2-1. find infiles Example	299
Figure 2-2. find insource Example.	301
Figure 3-1. UML Diagram for Components	778
Figure 3-2. UML Diagram for Comparator Classes	792
Figure 3-3. UML Diagram for Connectors	799
Figure 3-4. UML Diagram for Channels	830
Figure 3-5. UML Diagram for Reporting Classes	873

List of Tables

Table 1-1. Conventions for Command Syntax	15
Table 1-2. Examples of Object Names	20
Table 1-3. Wildcard Characters in HDL Object Names	23
Table 1-4. WildcardFilter Arguments	24
Table 1-5. WildcardFilter Argument Groups	26
Table 1-6. Keyboard Shortcuts for Command History	30
Table 1-7. VHDL Number Conventions: Style 1	31
Table 1-8. VHDL Number Conventions: Style 2	31
Table 1-9. Verilog Number Conventions	32
Table 1-10. Constants Supported for GUI Expresssions	35
Table 1-11. Array Constants Supported for GUI Expresssions	35
Table 1-12. Variables Supported for GUI Expresssions	35
Table 1-13. Array Variables Supported for GUI Expresssions	36
Table 1-14. Operators Supported for GUI Expresssions	37
Table 1-15. Casting Conversions Supported for GUI Expresssions	38
Table 1-16. VHDL Logic Values Used in GUI Search	42
Table 1-17. Verilog Logic Values Used in GUI Search	42
Table 2-1. Supported Commands	43
Table 2-2. Comparing Reference Objects to Test Objects	148
Table 2-3. Order and Type of Ranked Tests	216
Table 2-4. runStatus Command States	426
Table 2-5. runStatus -full Command Information	426
Table 2-6. Field Arguments for Window Searches	442
Table 2-7. Output Fields for tcheck_status Command	465
Table 2-8. Warning Message Categories for vcom -nowarn	532
Table 2-9. Order and Type of Ranked Tests	548
Table 2-10. Design Unit Properties	574
Table 2-11. Warning Message Categories for vlog -nowarn	631
Table 2-12. Warning Message Categories for vopt -nowarn	660
Table 2-13. Wave Window Commands for Cursor	718
Table 2-14. Wave Window Commands for Expanded Time Display	718
Table 2-15. Wave Window Commands for Controlling Display	719
Table 2-16. Wave Window Commands for Zooming	719
Table 3-1. Class Index	773
Table 3-2. Exports and Interfaces	800
Table 3-3. Interface Implementations	803
Table 3-4. Ports and Interfaces	804
Table 3-5. Deprecated Implementations	829

Chapter 1

Syntax and Conventions

Documentation Conventions

This manual uses the following conventions to define ModelSim™ command syntax.

Table 1-1. Conventions for Command Syntax

Syntax notation	Description
< >	angled brackets surrounding a syntax item indicate a user-defined argument; do not enter the brackets in commands
[]	square brackets generally indicate an optional item; if the brackets surround several words, all must be entered as a group; the brackets are not entered ¹
{ }	braces indicate that the enclosed expression contains one or more spaces yet should be treated as a single argument, or that the expression contains square brackets for an index; for either situation, the braces are entered
...	an ellipsis indicates items that may appear more than once; the ellipsis itself does not appear in commands
	the vertical bar indicates a choice between items on either side of it; do not include the bar in the command
monospaced type	monospaced type is used in command examples
#	comments included with commands are preceded by the number sign (#); useful for adding comments to DO files (macros)

1. One exception to this rule is when you are using Verilog syntax to designate an array slice. For example,

```
add wave {vector1[4:0]}
```

The square brackets in this case denote an index. The braces prevent the Tcl interpreter from treating the text within the square brackets as a Tcl command.

Note



Neither the prompt at the beginning of a line nor the <Enter> key that ends a line is shown in the command examples.

File and Directory Pathnames

Several ModelSim commands have arguments that point to files or directories. For example, the `-y` argument to `vlog` specifies the Verilog source library directory to search for undefined modules. Spaces in file pathnames must be escaped or the entire path must be enclosed in quotes. For example:

```
vlog top.v -y C:/Documents\ and\ Settings/projects/dut
```

or

```
vlog top.v -y "C:/Documents and Settings/projects/dut"
```

Design Object Names

Design objects are organized hierarchically. Each of the following objects creates a new level in the hierarchy:

- **VHDL** — component instantiation statement, block statement, and package
- **Verilog** — module instantiation, named fork, named begin, task and function
- **SystemVerilog** — class, package, program, and interface
- **SystemC** — module instantiation

Object Name Syntax

The syntax for specifying object names in ModelSim is as follows:

```
[<dataset_name><datasetSeparator>][<pathSeparator>][<hierarchicalPath>]  
<objectName>[<elementSelection>]
```

where

- **dataset_name** — is the name mapped to the WLF file in which the object exists. The currently active simulation is the “sim” dataset. Any loaded WLF file is referred to by the logical name specified when the WLF file was loaded. Refer to the chapter “[Recording Simulation Results With Datasets](#)” in the User’s Manual for more information.
- **datasetSeparator** — is the character used to terminate the dataset name. The default is colon (:), though a different character (other than backslash (\)) may be specified as the dataset separator via the [DatasetSeparator](#) variable in the `modelsim.ini` file. The default is a colon (:). This character must be different than the `pathSeparator` character.
- **pathSeparator** — is the character used to separate hierarchical object names. Normally, a backslash (/) is used for VHDL and a period (.) is used for Verilog, although other characters (except a backslash (\)) may be specified via the [PathSeparator](#) variable in

the *modelsim.ini* file. This character must be different than the `datasetSeparator`. Both (`.`) and forward slash (`/`) can be used for SystemC. Neither (`.`) nor (`/`) can be used when referring to the contents of a SystemVerilog package or class.

- **hierarchicalPath** — is a set of hierarchical instance names separated by a path separator and ending in a path separator prior to the `objectName`. For example, */top/proc/clock*.
- **objectName** — is the name of an object in a design.
- **elementSelection** — indicates some combination of the following:
 - **Array indexing** — Single array elements are specified using either parentheses (`()`) or square brackets (`[]`) around a single number.
 - **Array slicing** — Slices (or part-selects) of arrays are specified using either parentheses (`()`) or square brackets (`[]`) around a range specification. A range is two numbers separated by one of the following: " to ", " downto ", or a colon (`:`). See [Escaping Brackets and Spaces in Array Slices](#) for important information about using square brackets in ModelSim commands.
 - **Record field selection** — A record field is specified using a period (`.`) followed by the name of the field.
 - **C++ class, structure, and union member selection** — A class, structure, or union member is specified using the record field specification syntax, described just above.

SystemC Class, Structure, and Union Member Specification

You can specify members of SystemC structures and classes using HDL record syntax. The syntax for specifying members of a base class using ModelSim is different than C++. In C++, it is not necessary to specify the base class:

```
<instance>.<base_member>
```

Whereas, in ModelSim you *must* include the name of the base class:

```
<instance>.<base>.<base_member>
```

Example 1-1. Base- and Descendant-Class Specification

Let's say you have a base class and a descendant class:

```
class dog
{
    private:
    int value;
};
```

```
class beagle : public dog
{
    private:
        int value;
        dog d;
};
```

You have an `sc_signal<>` of type `beagle` somewhere in your code:

```
sc_signal<beagle> spot;
```

Legal names for viewing this signal are:

```
spot
spot.*
spot.value
spot.dog
spot.dog.*
spot.dog.value
```

Now, to examine the member *value* of the base class *dog*, you would type:

```
exa spot.dog.value
```

To examine the member *value* of member *d*, you would type:

```
exa spot.d.value
```

To examine the member *value*, you would type:

```
exa spot.value
```

SystemVerilog Scope Resolution Operator

SystemVerilog offers the scope resolution operator, double colon (`::`), for accessing classes within a package and static data within a class. The example below shows various methods of using this operator as well as alternatives using standard hierarchical references.

Example 1-2. SystemVerilog Scope Resolution Operator Example

```
package myPackage;
  class packet;
    static int a[0:1] = {1, 2};
    int b[0:1];
    int c;

    function new;
      b[0] = 3;
      b[1] = 4;
      c = a[0];
    endfunction
  endclass
endpackage : myPackage

module top;
  myPackage::packet my = new;
  int myint = my.a[1];
endmodule
```

The following [examine](#) examples access data from the class *packet*.

```
examine myPackage::packet::a
examine /top/my.a
```

Both of the above commands return the contents of the static array *a* within class *packet*.

```
examine myPackage::packet::a(0)
examine /top/my.a(0)
```

Both of the above commands return the contents of the first element of the static array *a* within class *packet*.

```
examine /top/my.b
```

Return the contents of the instance-specific array *b*.

```
examine /top/my.b(0)
```

Return the contents of the first element of the instance-specific array *b*.

When referring to the contents of a package or class, you cannot use the standard path separators, a period (`.`) or a forward slash (`/`).

Specifying Names

We distinguish between four "types" of object names: simple, relative, fully-rooted, and absolute.

- **Simple name** — does not contain any hierarchy. It is simply the name of an object (e.g., *clk* or *data[3:0]*) in the current context.

- **Relative name** — does not start with a path separator and may or may not include a dataset name or a hierarchical path (e.g., *u1/data* or *view:clk*). A relative name is relative to the current context in the current or specified dataset.
- **Fully-rooted name** — starts with a path separator and includes a hierarchical path to an object (e.g., */top/u1/clk*). There is a special case of a fully-rooted name where the top-level design unit name can be unspecified (e.g., */u1/clk*). In this case, the first top-level instance in the design is assumed.
- **Absolute name** — is an exactly specified hierarchical name containing a dataset name and a fully rooted name (e.g., *sim:/top/u1/clk*).

The current dataset is used when accessing objects where a dataset name is not specified as part of the name. The current dataset is determined by the dataset currently selected in the Structure window or by the last dataset specified in an [environment](#).

The current context in the current or specified dataset is used when accessing objects with relative or simple names. The current context is either the current process, if any, or the current instance if there is no current process or the current process is not in the current instance. The situation of the current process not being in the current instance can occur, for example, by selecting a different instance in the Structure tab or by using the [environment](#) to set the current context to a different instance.

[Table 1-2](#) contains examples of various ways of specifying object names.

Table 1-2. Examples of Object Names

Object Name	Description
<i>clk</i>	specifies the object <i>clk</i> in the current context
<i>/top/clk</i>	specifies the object <i>clk</i> in the top-level design unit.
<i>/top/block1/u2/clk</i>	specifies the object <i>clk</i> , two levels down from the top-level design unit
<i>block1/u2/clk</i>	specifies the object <i>clk</i> , two levels down from the current context
<i>array_sig[4]</i>	specifies an index of an array object
<i>{array_sig(1 to 10)}</i>	specifies a slice of an array object in VHDL or SystemC; see Escaping Brackets and Spaces in Array Slices for more information
<i>{mysignal[31:0]}</i>	specifies a slice of an array object in Verilog or SystemC; see Escaping Brackets and Spaces in Array Slices for more information
<i>record_sig.field</i>	specifies a field of a record, a C++ class or structure member, or a C++ base class

Escaping Brackets and Spaces in Array Slices

Because ModelSim is a Tcl-based tool, you must use curly braces (`{ }`) to "escape" square brackets and spaces when specifying array slices. For example:

```
toggle add {data[3:0]}
toggle add {data(3 to 0)}
```

For complete details on Tcl syntax, refer to [Tcl Command Syntax](#).

Further Details

As a Tcl-based tool, ModelSim commands follow Tcl syntax. One problem people encounter with ModelSim commands is the use of square brackets (`[]`) or spaces when specifying array slices. As shown on the previous page, square brackets are used to specify slices of arrays (e.g., `data[3:0]`). However, in Tcl, square brackets signify command substitution. Consider the following example:

```
set aluinputs [find -in alu/*]
```

ModelSim evaluates the **find** command first and then sets variable `aluinputs` to the result of the find command. Obviously you don't want this type of behavior when specifying an array slice, so you would use curly brace escape characters:

```
add wave {/s/abc/data_in[10:1]}
```

You must also use the escape characters if using VHDL syntax with spaces:

```
add wave {/s/abc/data_in(10 downto 1)}
```

Environment Variables and Pathnames

You can substitute environment variables for pathnames in any argument that requires a pathname. For example:

```
vlog -v $lib_path/und1
```

Assuming you have defined `$lib_path` on your system, `vlog` will locate the source library file `und1` and search it for undefined modules. Refer to [Environment Variables](#) for more information.

Name Case Sensitivity

Name case sensitivity is different for VHDL and Verilog. VHDL names are not case sensitive except for extended identifiers in VHDL 1076-1993 or later. In contrast, all Verilog names are case sensitive.

Names in ModelSim commands are case sensitive when matched against case sensitive identifiers, otherwise they are not case sensitive. SystemC names are case sensitive.

Extended Identifiers

The following are supported formats for extended identifiers for any command that takes an identifier.

```
{\ext ident!\ }  
  # Note that trailing space before closing brace is required  
  
\ext\ ident!\\  
  # All non-alpha characters escaped
```

Wildcard Characters

You can use wildcard characters in HDL object names for the following simulator commands:

- add dataflow
- add list
- add memory
- add watch
- add wave
- find
- log

When you execute any of these commands with a wildcard, the default behavior is to exclude the following object types:

- VHDL shared variables in packages and design units, constants, generics, and immediate assertions
- Verilog parameters, specparams, memories
- PSL and SystemVerilog assertions, covers, and endpoints
- Signals in cells

You can alter these exclusions with the `WildcardFilter` preference variable. Refer to the section [“Using the WildcardFilter Preference Variable”](#) for more information.

Table 1-3 identifies the supported wildcard characters.

Table 1-3. Wildcard Characters in HDL Object Names

Character Syntax	Description
*	matches any sequence of characters
?	matches any single character
[]	matches any one of the enclosed characters; a hyphen can be used to specify a range (for example, a-z, A-Z, 0-9); can be used <i>only</i> with the find command

Note

A wildcard character does not match a path separator. For example, `/dut/*` will match `/dut/signa` and `/dut/clk`. However, `/dut*` will not match either of those.

Using the WildcardFilter Preference Variable

The WildcardFilter preference variable controls which object types are excluded when performing wildcard matches with simulator commands.

Note

Your WildcardFilter settings are persistent from one invocation to the next.

Procedure

Determining the Current WildcardFilter Variable Settings

Enter the following command:

```
set WildcardFilter
```

which returns a list of currently set arguments for exclusion.

Changing the WildcardFilter Settings from the Command Line

Refer to the list of WildcardFilter arguments in [Table 1-4](#) to determine what you want to exclude from wildcard matches, then enter the following command:

```
set WildcardFilter "<argument> ..."
```

Note that you must enclose the space-separated list of arguments in quotation marks.

Changing the WildcardFilter Settings back to the Default

Enter the following command:

set WildcardFilter default

Changing the WildcardFilter settings from the GUI

1. Choose **Tools > Wildcard Filter** from the main menu.
2. Select the individual Filters you want to exclude from wildcard searches ([Table 1-4](#) describes each option), or select Composite Filters to activate related filters ([Table 1-5](#) describes each composite option).
3. Click OK.

WildcardFilter Argument Descriptions

[Table 1-4](#) provides a list of the WildcardFilter arguments.

Table 1-4. WildcardFilter Arguments

Argument	Description
Alias	VHDL Alias
Architecture	VHDL Architecture
Assertion	Concurrent SystemVerilog or PSL assertion
Block	VHDL or Verilog block
CellInternal	Signals in cells, where a cell is defined as 1) a module within a 'celldefine 2) a Verilog module found with a library search (using either vlog -v or vlog -y) and compiled with vlog +libcell or 3) a module containing a specify block
Class	Verilog class declaration
ClassReference	SystemVerilog class reference
ClockingBlock	Verilog clocking block
Compare	Waveform comparison signal
Configuration	Verilog configuration
Constant	VHDL constant
Cover	SystemVerilog or PSL cover statements
Covergroup	SystemVerilog or PSL covergroup
Coverpoint	Verilog coverpoint
Cross	Verilog cross
Endpoint	SystemVerilog assertion objects created for sequences on which the method "ended/triggered" is used. PSL assertion objects created for sequences for which the builtin function "ended()" is used.

Table 1-4. WildcardFilter Arguments

Argument	Description
Foreign	VHDL foreign
Function	Verilog function
Generate	VHDL generate
Generic	VHDL generic
ImmediateAssert	VHDL immediate assertions
Integer	VHDL integer
Interface	SystemVerilog interface
Memory	Verilog memories
Module	Verilog module
NamedEvent	Verilog named event
Net	Verilog net
Package	VHDL package
ParamClass	Verilog parameterized class
Parameter	Verilog parameter
Port	Verilog port
Primitive	Verilog primitive
Process	VHDL process
Property	Assertion property
Real	Verilog real registers
Reg	Verilog register
Root	All objects
ScExport	SystemC export
ScHierChannel	SystemC hierarchical channel
ScMethod	SystemC method
ScModule	SystemC module
ScPort	SystemC port
ScPrimChannel	SystemC primitive channel
ScThread	SystemC thread
ScVariable	SystemC variable
Sequence	SystemVerilog sequence

Table 1-4. WildcardFilter Arguments

Argument	Description
Signal	VHDL signal
SpecParam	Verilog specparam
Statement	Verilog statement
Task	Verilog task
TaThreadMon	Assertion thread monitor object
Time	Verilog time registers
TrStream	Transaction stream
TrStreamArray	Transaction stream array
Variable	VHDL shared variables in packages and design units.
VirtualExpr	Virtual expression
VirtualRegion	Virtual region
VirtualSignal	Virtual signal
vlGenerateBlock	Verilog generate block
vlPackage	Verilog package
vlProgram	Verilog program
vlTypedef	Verilog typedef

Table 1-5 provides a list of the group aliases of WildcardFilter arguments.

Table 1-5. WildcardFilter Argument Groups

Group Argument	Specific arguments included
AllVHDLScopes	Architecture, Block, Generate, Package, Foreign
AllVHDL	Architecture, Block, Generate, Package, Foreign, Process, Signal, Variable, Constant, Generic, Alias
AllVerilogScopes	Block, vlGenerateBlock, Module, Task, Function, Statement, Class, Cross, Covergroup, Coverpoint, vlPackage, vlTypedef, ParamClass, ClockingBlock
AllVerilogVars	Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, ClassReference
AllVerilog	Block, vlGenerateBlock, Module, Primitive, Task, Function, Statement, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, Class, Cross, Covergroup, Coverpoint, vlPackage, vlTypedef, ParamClass, ClockingBlock, ClassReference

Table 1-5. WildcardFilter Argument Groups

Group Argument	Specific arguments included
VirtualSignals	VirtualSignal, VirtualExpr
Virtual	VirtualRegion, VirtualSignal, VirtualExpr
SystemCSignals	ScPrimChannel, ScPort, ScExport
SystemCProcess	ScMethod, ScThread
SystemC	ScModule, ScPrimChannel, ScVariable, ScPort, ScMethod, ScThread, ScExport, ScHierChannel
TR	TrStream, TrStreamArray
AllHDLScopes	Architecture, Block, Generate, Package, Foreign, vlGenerateBlock, Module, Task, Function, Statement, Class, Cross, Covergroup, Coverpoint, vlPackage, vlTypedef, ParamClass, ClockingBlock
AllHDLSignals	Signal, Variable, Constant, Generic, Alias, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, VirtualSignal, VirtualExpr, ClassReference
AllVariables	Variable, Constant, Generic, Alias, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, ClassReference
AllHDLSignalsVars	Signal, Variable, Constant, Generic, Alias, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, VirtualSignal, VirtualExpr, ClassReference
AllHDL	Architecture, Block, Generate, Package, Foreign, vlGenerateBlock, Signal, Variable, Module, Task, Function, Statement, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, VirtualSignal, VirtualExpr, Class, Cross, Covergroup, Coverpoint, vlPackage, vlTypedef, ParamClass, ClockingBlock, ClassReference
AllScopes	Architecture, Block, Generate, Package, Foreign, vlGenerateBlock, Module, Task, Function, Statement, VirtualRegion, ScModule, Class, Cross, Covergroup, Coverpoint, vlPackage, vlTypedef, ParamClass, ClockingBlock
AllSignals	Signal, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, VirtualSignal, VirtualExpr, ScPrimChannel, Endpoint, ScPort, TrStream, TrStreamArray, ScExport, ClassReference

Table 1-5. WildcardFilter Argument Groups

Group Argument	Specific arguments included
AllSignalsVars	Signal, Variable, Constant, Generic, Alias, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, VirtualSignal, VirtualExpr, ScPrimChannel, Endpoint, ScVariable, ScPort, TrStream, TrStreamArray, ScExport, ClassReference
AllConstants	Constant, Generic, Parameter, SpecParam
AllProcesses	Process, ScMethod, ScThread
Default	Variable, Constant, Generic, Parameter, SpecParam, Memory, Assertion, Cover, Endpoint, ScVariable, CellInternal, ImmediateAssert

Simulator Variables

ModelSim variables can be referenced in simulator commands by preceding the name of the variable with the dollar sign (\$) character. ModelSim uses global variables for simulator state variables, simulator control variables, simulator preference variables, and user-defined variables. Refer to [modelsim.ini Variables](#) in the User's Manual for more information on variables.

The [report](#) command returns a list of current settings for either the simulator state or simulator control variables.

Simulation Time Units

You can specify the time unit for delays in all simulator commands that have time arguments. For example:

```
force clk 1 50 ns, 1 100 ns -repeat 1 us
run 2 ms
```

Note that all the time units in a ModelSim command need not be the same.

Unless you specify otherwise as in the examples above, simulation time is always expressed using the resolution units that are specified by the [UserTimeUnit](#) variable.

By default, the specified time units are assumed to be relative to the current time unless the value is preceded by the character @, which signifies an absolute time specification.

Argument Files

You can load additional arguments into some commands by using argument files, which are specified with the `-f` argument. The following commands support the `-f` argument:

```
sccom  vlog  vcom  vencrypt  vmake  vopt  vsim
```

The `-f <filename>` argument specifies a file that contains additional command line arguments. The following sections outline some syntax rules for argument files.

- **Single Quotes (‘ ‘)**— allows you to group arbitrary characters so that no character substitution occurs within the quotes, such as environment variable expansion or escaped characters.

```
+acc=rn+'\mymodule'  
//does not treat the '\ ' as an escape character
```

- **Double Quotes (“ ”)**— allows you to group arbitrary characters so that Tcl-style backslash substitution and environment variable expansion is performed.

```
+acc=rn+"\\mymodule\\$VAR"  
// escapes the path separators ( \ ) and substitutes  
// your value of '$VAR'
```

- **Unquoted** — the following are notes on what occurs when some information is not quoted:

- **Tcl backslash substitution** — any unquoted backslash (\) will be treated as an escape character.

```
+acc=rn\\mymodule  
// the leading '\ ' is considered an escape character
```

- **Environment variable expansion** — any unquoted environment variable, such as `$envname`, will be expanded. You can also use curly braces ({ }) in your environment variable, such as `${envname}`.

```
+acc=rn\\$MODULE  
// the leading '\ ' is considered an escape character and the  
// variable $MODULE is expanded
```

- **Newline Character** — you can specify arguments on separate lines in the argument file, with the newline characters treated as space characters. There is no need to put a backslash (\) at the end of each line.
- **Comments** — Comments within the argument files follow these rules:
 - All text in a line beginning with `//` to its end is treated as a comment.
 - All text bracketed by `/* ... */` is treated as a comment.

Command Shortcuts

- You may abbreviate command syntax, but there's a catch — the minimum number of characters required to execute a command are those that make it unique. Remember, as we add new commands some of the old shortcuts may not work. For this reason ModelSim does not allow command name abbreviations in macro files. This minimizes your need to update macro files as new commands are added.
- Multiple commands may be entered on one line if they are separated by semi-colons (;). For example:

```
ModelSim> vlog -nodebug=ports level3.v level2.v ; vlog -nodebug top.v
```

The return value of the last function executed is the only one printed to the transcript. This may cause some unexpected behavior in certain circumstances. Consider this example:

```
vsim -c -do "run 20 ; simstats ; quit -f" top
```

You probably expect the **simstats** results to display in the Transcript window, but they will not, because the last command is **quit -f**. To see the return values of intermediate commands, you must explicitly print the results. For example:

```
vsim -do "run 20 ; echo [simstats]; quit -f" -c top
```

Command History Shortcuts

You can review simulator command history or rerun previous commands by using keyboard shortcuts at the ModelSim/VSIM prompt. [Table 1-6](#) contains a list of these shortcuts.

Table 1-6. Keyboard Shortcuts for Command History

Shortcut	Description
!!	repeats the last command
!n	repeats command number n; n is the VSIM prompt number (e.g., for this prompt: VSIM 12>, n =12)
!abc	repeats the most recent command starting with "abc"
^xyz^ab^	replaces "xyz" in the last command with "ab"
up and down arrows	scrolls through the command history with the keyboard arrows
click on prompt	left-click once on a previous ModelSim or VSIM prompt in the transcript to copy the command typed at that prompt to the active cursor
his or history	shows the last few commands (up to 50 are kept)

Numbering Conventions

Numbers in ModelSim can be expressed in either VHDL or Verilog style. You can use two styles for VHDL numbers and one for Verilog.

VHDL Numbering Conventions

There are two types of VHDL number styles:

VHDL Style 1

```
[ - ] [ radix # ] value [ # ]
```

Table 1-7. VHDL Number Conventions: Style 1

Element	Description
-	indicates a negative number; optional
radix	can be any base in the range 2 through 16 (2, 8, 10, or 16); by default, numbers are assumed to be decimal; optional
value	specifies the numeric value, expressed in the specified radix; required
#	is a delimiter between the radix and the value; the first # sign is required if a radix is used, the second is always optional

A '-' can also be used to designate a "don't care" element when you search for a signal value or expression in the List or Wave window. If you want the '-' to be read as a "don't care" element, rather than a negative sign, be sure to enclose the number in double quotes. For instance, you would type "-0110--" as opposed to -0110--. If you don't include the double quotes, ModelSim will read the '-' as a negative sign. For example:

```
16#FFca23#
2#11111110
-23749
```

VHDL Style 2

```
base "value"
```

Table 1-8. VHDL Number Conventions: Style 2

Element	Description
base	specifies the base; binary: B, octal: O, hex: X; required
"value"	specifies digits in the appropriate base with optional underscore separators; default is decimal; required

For example:

```
B"11111110"  
X"FFca23"
```

Searching for VHDL Arrays in the Wave and List Windows

Searching for signal values in the Wave or List window may not work correctly for VHDL arrays if the target value is in decimal notation. You may get an error that the value is of incompatible type. Since VHDL does not have a radix indicator for decimal, the target value may get misinterpreted as a scalar value. Prefixing the value with the Verilog notation 'd should eliminate the problem, even if the signal is VHDL.

Verilog Numbering Conventions

Verilog numbers are expressed in the style:

```
[ - ] [ size ] [ base ] value
```

Table 1-9. Verilog Number Conventions

Element	Description
-	indicates a negative number; optional
size	the number of bits in the number; optional
base	specifies the base; binary: 'b or 'B, octal: 'o or 'O, decimal: 'd or 'D, hex: 'h or 'H; optional
value	specifies digits in the appropriate base with optional underscore separators; default is decimal; required

A '-' can also be used to designate a "don't care" element when you search for a signal value or expression in the List or Wave windows. If you want the '-' to be read as a "don't care" element, rather than a negative sign, be sure to enclose the number in double quotes. For instance, you would type "-0110--" as opposed to 7'b-0110--. If you don't include the double quotes, ModelSim will read the '-' as a negative sign. For example:

```
'b11111110          8'b11111110  
'Hffca23           21'H1fca23  
-23749
```

GUI_expression_format

The GUI_expression_format is an option of several simulator commands that operate within the ModelSim GUI environment. The expressions help you locate and examine objects within the List and Wave windows (expressions may also be used through the **Edit > Search** menu in both windows). The commands that use the expression format are:

compare add, compare clock, compare configure, configure, examine, searchlog, virtual function, virtual signal, down, left, right, up.

Expression Typing

GUI expressions are typed. The supported types consist of the following scalar and array types.

Scalar Types

The scalar types are as follows: boolean, integer, real, time (64-bit integer), enumeration, and signal state. Signal states are represented by the nine VHDL `std_logic` states: 'U' 'X' '0' '1' 'Z' 'W' 'L' 'H' and '-'.

Verilog states 0, 1, x, and z are mapped into these states and the Verilog strengths are ignored. Conversion is done automatically when referencing Verilog nets or registers.

SystemC scalar types supported are: all the C/C++ types except class, structure, union, and array, as well as SystemC types `sc_logic` and `sc_bit`.

Array Types

The supported array types are signed and unsigned arrays of signal states. This would correspond to the VHDL `std_logic_array` type. Verilog registers are automatically converted to these array types. The array type can be treated as either UNSIGNED or SIGNED, as in the IEEE `std_logic_arith` package. Normally, referencing a signal array causes it to be treated as UNSIGNED by the expression evaluator; to cause it to be treated as SIGNED, use casting as described below. Numeric operations supported on arrays are performed by the expression evaluator via ModelSim's built-in `numeric_standard` (and similar) package routines. The expression evaluator selects the appropriate numeric routine based on SIGNED or UNSIGNED properties of the array arguments and the result.

The enumeration types supported are any VHDL enumerated type. Enumeration literals may be used in the expression as long as some variable of that enumeration type is referenced in the expression. This is useful for sub-expressions of the form:

```
(/memory/state == reading)
```

The supported SystemC aggregate types are the C/C++ array types: union, class, structure, and array. Also supported are the SystemC array types: `sc_bv<w>`, `sc_lv<w>`, `sc_int<w>`, etc.

Expression Syntax

GUI expressions generally follow C-language syntax, with both VHDL-specific and Verilog-specific conventions supported. These expressions are not parsed by the Tcl parser, and so do not support general Tcl; parentheses should be used rather than braces. Procedure calls are not supported.

A GUI expression can include the following elements: Tcl macros, constants, array constants, variables, array variables, signal attributes, operators, and casting.

Tcl Macros

Macros are useful for pre-defined constants or for entire expressions that have been previously saved. The substitution is done only once, when the expression is first parsed. Macro syntax is:

`$<name>`

Substitutes the string value of the Tcl global variable <name>.

Constants

Table 1-10. Constants Supported for GUI Expressions

Type	Values
boolean value	true false TRUE FALSE
integer	[0-9]+
real number	<int> (<int>.<int>[exp]) where the optional [exp] is: (e E)[+ -][0-9]+
time	integer or real optionally followed by time unit
enumeration	VHDL user-defined enumeration literal
single bit constants	expressed as any of the following: 0 1 x X z Z U H L W 'U' 'X' '0' '1' 'Z' 'W' 'L' 'H' '-' 1'b0 1'b1

Array Constants, Expressed in Any of the Following Formats

Table 1-11. Array Constants Supported for GUI Expressions

Type	Values
VHDL # notation	<int>#<alphanum>[#] Example: 16#abc123#
VHDL bitstring	"(U X 0 1 Z W L H -)*" Example: "11010X11"
Verilog notation	[-][<int>]'(b B o O d D h H) <alphanum> (where <alphanum> includes 0-9, a-f, A-F, and '-')
Based notation	0x..., 0X..., 0o..., 0O..., 0b..., 0B... ModelSim automatically zero fills unspecified upper bits.

Variables

Table 1-12. Variables Supported for GUI Expressions

Variable	Type
Name of a signal	The name may be a simple name, a VHDL or Verilog style extended identifier, or a VHDL or Verilog style path. The signal must be one of the following types: -- VHDL signal of type INTEGER, REAL, or TIME -- VHDL signal of type std_logic or bit -- VHDL signal of type user-defined enumeration -- Verilog net, Verilog register, Verilog integer, or Verilog real -- SystemC primitive channels of type scalar (e.g. bool, int, etc.)

Table 1-12. Variables Supported for GUI Expressions

Variable	Type
NOW	Returns the value of time at the current location in the WLF file as the WLF file is being scanned (not the most recent simulation time).

Array variables

Table 1-13. Array Variables Supported for GUI Expressions

Variable	Type
Name of a signal	-- VHDL signals of type bit_vector or std_logic_vector -- Verilog register -- Verilog net array -- SystemC primitive channels of type vector (e.g. sc_bv, sc_int, etc.) A subrange or index may be specified in either VHDL or Verilog syntax. Examples: mysignal(1 to 5), mysignal[1:5], mysignal (4), mysignal [4]

Signal attributes

```
<name>'event  
<name>'rising  
<name>'falling  
<name>'delayed(  
<name>'hasX
```

The 'delayed attribute lets you assign a delay to a VHDL signal. To assign a delay to a signal in Verilog, use “#” notation in a sub-expression (e.g., #-10 /top/signalA).

The hasX attribute lets you search for signals, nets, or registers that contains an X (unknown) value.

See [Examples of Expression Syntax](#) below for further details on 'delayed and 'hasX.

Operators

Table 1-14. Operators Supported for GUI Expressions

Operator	Description	Operator	Description
&&	boolean and	sll/SLL	shift left logical
	boolean or	sla/SLA	shift left arithmetic
!	boolean not	srl/SRL	shift right logical
==	equal	sra/SRA	shift right arithmetic
!=	not equal	ror/ROR	rotate right
===	exact equal ¹	rol/ROL	rotate left
!==	exact not equal ¹	+	arithmetic add
<	less than	-	arithmetic subtract
<=	less than or equal	*	arithmetic multiply
>	greater than	/	arithmetic divide
>=	greater than or equal	mod/MOD	arithmetic modulus
not/NOT/~	unary bitwise inversion	rem/REM	arithmetic remainder
and/AND	bitwise and	<vector_expr>	OR reduction
nand/NAND	bitwise nand	^<vector_expr>	XOR reduction
or/OR/	bitwise or		
nor/NOR	bitwise nor		
xor/XOR	bitwise xor		
xnor/XNOR	bitwise xnor		

1. This operator is allowed to be compatible with other simulators.

Note



Arithmetic operators use the std_logic_arith package.

Casting

Table 1-15. Casting Conversions Supported for GUI Expressions

Casting	Description
(bool)	convert to boolean
(boolean)	convert to boolean
(int)	convert to integer
(integer)	convert to integer
(real)	convert to real
(time)	convert to 64-bit integer
(std_logic)	convert to 9-state signal value
(signed)	convert to signed vector
(unsigned)	convert to unsigned vector
(std_logic_vector)	convert to unsigned vector

Examples of Expression Syntax

```
/top/bus & $bit_mask
```

This expression takes the bitwise AND function of signal */top/bus* and the array constant contained in the global Tcl variable *bit_mask*.

```
clk'event && (/top/xyz == 16'hffae)
```

This expression evaluates to a boolean true when signal *clk* changes and signal */top/xyz* is equal to hex *ffae*; otherwise is false.

```
clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)
```

Evaluates to a boolean true when signal *clk* just changed from low to high and signal *mystate* is the enumeration *reading* and signal */top/u3/addr* is equal to the specified 32-bit hex constant; otherwise is false.

```
(/top/u3/addr and 32'hff000000) == 32'hac000000
```

Evaluates to a boolean true when the upper 8 bits of the 32-bit signal */top/u3/addr* equals hex *ac*.

```
/top/signalA'delayed(10ns)
```

This expression returns */top/signalA* delayed by 10 ns.

```
/top/signalA'delayed(10 ns) && /top/signalB
```

This expression takes the logical AND of a delayed */top/signalA* with */top/signalB*.

```
virtual function { (#-10 /top/signalA) && /top/signalB}  
mySignalB_AND_DelayedSignalA
```

This evaluates */top/signalA* at 10 simulation time steps before the current time, and takes the logical AND of the result with the current value of */top/signalB*. The '#' notation uses positive numbers for looking into the future, and negative numbers for delay. This notation does not support the use of time units.

```
((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode == writing)
```

Evaluates to a boolean true when WLF file time is between 23 and 54 microseconds, *clk* just changed from low to high, and signal mode is enumeration writing.

```
searchlog -expr {dbus'hasX} {0 ns} dbus
```

Searches for an 'X' in *dbus*. This is equivalent to the expression: *{dbus(0) == 'x' // dbus(1) == 'x'} . . .* This makes it possible to search for X values without having to write a type specific literal.

Signal and Subelement Naming Conventions

ModelSim supports naming conventions for VHDL and Verilog signal pathnames, VHDL array indexing, Verilog bit selection, VHDL subrange specification, and Verilog part selection. All supported naming conventions for VHDL and Verilog are valid for SystemC designs.

Examples in Verilog and VHDL syntax:

```
top.chip.vlogsig  
/top/chip/vhdlsig  
vlogsig[3]  
vhdlsig(9)  
vlogsig[5:2]  
vhdlsig(5 downto 2)
```

Grouping and Precedence

Operator precedence generally follows that of the C language, but we recommend liberal use of parentheses.

Concatenation of Signals or Subelements

Elements in the concatenation that are arrays are expanded so that each element in the array becomes a top-level element of the concatenation. But for elements in the concatenation that are records, the entire record becomes one top-level element in the result. To specify that the records be broken down so that their subelements become top-level elements in the concatenation, use the **concat_flatten** directive. Currently we do not support leaving full arrays as elements in the result. (Please let us know if you need that option.)

If the elements being concatenated are of incompatible base types, a VHDL-style record will be created. The record object can be expanded in the Objects and Wave windows just like an array of compatible type elements.

Concatenation Syntax for VHDL

```
<signalOrSliceName1> & <signalOrSliceName2> & ...
```

Concatenation Syntax for Verilog

```
&{<signalOrSliceName1>, <signalOrSliceName2>, ... }  
&{<count>{<signalOrSliceName1>}, <signalOrSliceName2>, ... }
```

Note that the concatenation syntax begins with "&{" rather than just "{". Repetition multipliers are supported, as illustrated in the second line. The repetition element itself may be an arbitrary concatenation subexpression.

Concatenation Directives

A concatenation directive (as illustrated below) can be used to constrain the resulting array range of a concatenation or influence how compound objects are treated. By default, the concatenation will be created with a descending index range from $(n-1)$ downto 0, where n is the number of elements in the array.

```
(concat_range 31:0)<concatenationExpr> # Verilog syntax  
(concat_range (31:0))<concatenationExpr> # Also Verilog syntax  
(concat_range (31 downto 0))<concatenationExpr> # VHDL syntax
```

The **concat_range** directive completely specifies the index range.

```
(concat_ascending) <concatenationExpr>
```

The **concat_ascending** directive specifies that the index start at zero and increment upwards.

```
(concat_flatten) <concatenationExpr>
```

The **concat_flatten** directive flattens the signal structure hierarchy.

```
(concat_noflatten) <concatenationExpr>
```

The **concat_noflatten** directive groups signals together without merging them into one big array. The signals become elements of a record and retain their original names.

When expanded, the new signal looks just like a group of signals. The directive can be used hierarchically with no limits on depth.

```
(concat_sort_wild_ascending) <concatenationExpr>
```

The **concat_sort_wild_ascending** directive gathers signals by name in ascending order (the default is descending).


```
(concat_reverse) <concatenationExpr>
```

The **concat_reverse** directive reverses the bits of the concatenated signals.

Examples of Concatenation

```
&{ "mybusbasename*" }
```

Gathers all signals in the current context whose names begin with "mybusbasename", sorts those names in descending order, and creates a bus with index range ($n-1$) downto 0, where n is the number of matching signals found. (Note that it currently does not derive the index name from the tail of the one-bit signal name.)

```
(concat_range 13:4)&{ "mybusbasename*" }
```

Specifies the index range to be 13 downto 4, with the signals gathered by name in descending order.

```
(concat_ascending)&{ "mybusbasename*" }
```

Specifies an ascending range of 0 to $n-1$, with the signals gathered by name in descending order.

```
(concat_ascending)((concat_sort_wild_ascending)&{ "mybusbasename*" })
```

Specifies an ascending range of 0 to $n-1$, with the signals gathered by name in ascending order.

```
(concat_reverse)(bus1 & bus2)
```

Specifies that the bits of bus1 and bus2 be reversed in the output virtual signal.

Record Field Members

Arbitrarily-nested arrays and records are supported, but operators will only operate on one field at a time. That is, the expression $\{a == b\}$ where a and b are records with multiple fields, is not supported. This would have to be expressed as:

```
{(a.f1 == b.f1) && (a.f2 == b.f2) ...}
```

Examples:

```
vhdsig.field1  
vhdsig.field1.subfield1  
vhdsig.(5).field3  
vhdsig.field4(3 downto 0)
```

Searching for Binary Signal Values in the GUI

When you use the GUI to search for signal values displayed in 4-state binary radix, you should be aware of how ModelSim maps between binary radix and `std_logic`. The issue arises because

there is no “un-initialized” value in binary, while there is in std_logic. So, ModelSim relies on mapping tables to determine whether a match occurs between the displayed binary signal value and the underlying std_logic value.

This matching algorithm applies only to searching using the GUI. It does not apply to VHDL or Verilog test benches.

For comparing VHDL std_logic/std_ulogic objects, ModelSim uses the table shown below. An entry of “0” in the table is “no match”; an entry of “1” is a “match”; an entry of “2” is a match only if you set the Tcl variable **STDLOGIC_X_MatchesAnything** to 1. Note that X will match a U, and - will match anything.

Table 1-16. VHDL Logic Values Used in GUI Search

Search Entry	Matches as follows:								
	U	X	0	1	Z	W	L	H	-
U	1	1	0	0	0	0	0	0	1
X	1	1	2	2	2	2	2	2	1
0	0	2	1	0	0	0	1	0	1
1	0	2	0	1	0	0	0	1	1
Z	0	2	0	0	1	0	0	0	1
W	0	2	0	0	0	1	0	0	1
L	0	2	1	0	0	0	1	0	1
H	0	2	0	1	0	0	0	1	1
-	1	1	1	1	1	1	1	1	1

For comparing Verilog net values, ModelSim uses the table shown below. An entry of “2” is a match only if you set the Tcl variable “VLOG_X_MatchesAnything” to 1.

Table 1-17. Verilog Logic Values Used in GUI Search

Search Entry	Matches as follows:			
	0	1	Z	X
0	1	0	0	2
1	0	1	0	2
Z	0	0	1	2
X	2	2	2	1

This table also applies to SystemC types: sc_bit, sc_bv, sc_logic, sc_int, sc_uint, sc_bigint, sc_biguint.

Chapter 2

Commands

This chapter describes ModelSim commands that you can enter either on the command line of the Main window or in a macro file. Some commands are automatically entered on the command line when you use the graphical user interface.

Note that, in addition to the simulation commands listed in this chapter, you can also use the Tcl commands described in the Tcl man pages (use the Main window menu selection: **Help > Tcl Man Pages**).

[Table 2-1](#) provides a brief description of each ModelSim command. For more information on command details, arguments, and examples, click the link in the Command name column.

Table 2-1. Supported Commands

Command name	Action
.main clear	clears the Main window transcript
abort	halts the execution of a macro file interrupted by a breakpoint or error
add atv	opens an Assertion Thread View (ATV) window for the specified assert or cover directive (designated by its pathname), at the specified evaluation attempt start time
add button	adds a user-defined button to the Main window button bar
add dataflow	adds the specified object(s) to the Dataflow window
add list	lists VHDL signals and variables, and Verilog nets and registers, and their values in the List window
add log	also known as the log command; see log
add memory	opens the specified memory in the MDI frame of the Main window
add schematic	adds the specified object(s) to the Schematic window
add testbrowser	adds .ucdb files to the Test Management Browser
add watch	adds signals or variables to the Watch window
add wave	adds VHDL signals and variables, and Verilog nets and registers to the Wave window
add_cmdhelp	adds an entry to the command-line help; use the help command to display the help text

Table 2-1. Supported Commands (cont.)

Command name	Action
<code>add_menu</code>	adds a menu to the menu bar of the specified window, using the specified menu name
<code>add_menubc</code>	creates a checkbox within the specified menu of the specified window
<code>add_menuitem</code>	creates a menu item within the specified menu of the specified window
<code>add_separator</code>	adds a separator as the next item in the specified menu path in the specified window
<code>add_submenu</code>	creates a cascading submenu within the specified menu path of the specified window
<code>alias</code>	creates a new Tcl procedure that evaluates the specified commands
<code>assertion action</code>	allows you to set the assertion action for assertion starts, fails, passes, or antecedent matches
<code>assertion active</code>	instructs the simulator to report on any active assertion directives at the end of simulation (EOS)
<code>assertion count</code>	returns the sum of the assertion failure counts for the specified set of assertion directive instances
<code>assertion enable</code>	enables and disables assertions and cover directives.
<code>assertion fail</code>	configures fail tracking for SystemVerilog and PSL assertions
<code>assertion pass</code>	configures pass tracking for SystemVerilog and PSL assertions
<code>assertion profile</code>	generates a fine grained profile of memory usage for assertions and cover directives
<code>atv log</code>	enables or disables assertion thread viewing (ATV) for the specified assertion(s)
<code>batch_mode</code>	returns a 1 if ModelSim is operating in batch mode, otherwise returns a 0
<code>bd</code>	deletes a breakpoint
<code>bookmark add wave</code>	adds a bookmark to the specified Wave window
<code>bookmark delete wave</code>	deletes bookmarks from the specified Wave window
<code>bookmark goto wave</code>	zooms and scrolls a Wave window using the specified bookmark
<code>bookmark list wave</code>	displays a list of available bookmarks
<code>bp</code>	sets a breakpoint
<code>cd</code>	changes the ModelSim local directory to the specified directory

Table 2-1. Supported Commands (cont.)

Command name	Action
<code>cdbg</code>	provides command-line equivalents of the menu options that are available for C Debug .
<code>change</code>	modifies the value of a VHDL variable or Verilog register variable
<code>change_menu_cmd</code>	changes the command to be executed for a specified menu item label, in the specified menu, in the specified window
<code>check contention add</code>	enables contention checking for the specified nodes
<code>check contention config</code>	writes checking messages to a file
<code>check contention off</code>	disables contention checking for the specified nodes
<code>check float add</code>	enables float checking for the specified nodes
<code>check float config</code>	writes checking messages to a file
<code>check float off</code>	disables float checking for the specified nodes
<code>check stable off</code>	disables stability checking
<code>check stable on</code>	enables stability checking on the entire design
<code>checkpoint</code>	saves the state of your simulation
<code>compare add</code>	compares signals in a reference design against signals in a test design
<code>compare annotate</code>	marks a compare difference as "ignore" or tags it with a text message
<code>compare clock</code>	defines a clock to be used with clocked-mode comparisons
<code>compare configure</code>	modifies options for compare signals or regions
<code>compare continue</code>	continues difference computation that had been suspended
<code>compare delete</code>	deletes a signal or region from the current comparison
<code>compare end</code>	closes the currently open comparison
<code>compare info</code>	lists the results of the comparison
<code>compare list</code>	lists all the compare add commands currently in effect
<code>compare options</code>	sets defaults for options used in other compare commands
<code>compare reload</code>	reloads a comparison previously saved with the <code>compare savediffs</code> command
<code>compare reset</code>	clears the current compare differences
<code>compare run</code>	runs the comparison on selected signals
<code>compare savediffs</code>	saves comparison differences to a file that can be reloaded later

Table 2-1. Supported Commands (cont.)

Command name	Action
<code>compare saverules</code>	saves comparison setup information to a file that can be reloaded later
<code>compare see</code>	displays a comparison difference in the Wave window
<code>compare start</code>	starts a new dataset comparison
<code>compare stop</code>	halts active difference computation
<code>compare update</code>	updates the comparison differences
<code>configure</code>	invokes the List or Wave widget configure command for the current default List or Wave window
<code>context</code>	provides several operations on a context's name
<code>coverage analyze</code>	displays test oriented information based on a <code>vcover merge</code> operation
<code>coverage attribute</code>	displays attributes in the currently loaded database
<code>coverage clear</code>	clears all coverage data obtained during previous run commands. Undocumented for 6.4
<code>coverage edit</code>	opens a coverage dataset (.ucdb) to edit the contents. Used only in Coverage View mode (<code>vsim -viewcov</code>)
<code>coverage exclude</code>	loads an exclusion filter file; or, allows you to exclude specific objects, files, design units, lines in a source file or rows within a table, etc.
<code>coverage goal</code>	Sets the value of UCDB-wide goals
<code>coverage open</code>	Opens UCDB datasets for viewing in the GUI in Coverage View mode
<code>coverage ranktest</code>	ranks coverage data according to user-specified tests
<code>coverage report</code>	produces a textual output of the coverage statistics that have been gathered up to this point
<code>coverage save</code>	saves current coverage statistics to a file that can be reloaded later, preserving instance-specific information
<code>coverage tag</code>	adds or removes tags from specified objects
<code>coverage testnames</code>	displays test names in the current UCDB file loaded
<code>coverage unlinked</code>	reports on test plan item with no coverage associated, or design coverage not associated with a test plan item
<code>coverage weight</code>	sets a global per-type weight for total coverage calculations
<code>dataset alias</code>	assigns an additional name to a dataset

Table 2-1. Supported Commands (cont.)

Command name	Action
dataset clear	clears the current simulation WLF file
dataset close	closes a dataset
dataset config	configures WLF file settings after dataset is open
dataset info	reports information about the specified dataset
dataset list	lists the open dataset(s)
dataset open	opens a dataset and references it by a logical name
dataset rename	changes the logical name of an opened dataset
dataset restart	unloads specified or current dataset
dataset save	saves data from the current WLF file to a specified file
dataset snapshot	saves data from the current WLF file at a specified interval
delete	removes objects from either the List or Wave window
describe	displays information about the specified HDL object
disablebp	turns off breakpoints and when commands
disable_menu	disables the specified menu within the specified window
disable_menuitem	disables the specified menu item within the specified menu path of the specified window
do	executes commands contained in a macro file
down	searches for signal transitions or values in the specified List window
drivers	displays in the Main window the current value and scheduled future values for all the drivers of a specified VHDL signal or Verilog net
dumplog64	dumps the contents of the <i>vsim.wlf</i> file in a readable format
echo	displays a specified message in the Main window
edit	invokes the editor specified by the EDITOR environment variable
enablebp	turns on breakpoints and when commands turned off by the disablebp command
enable_menu	enables a previously-disabled menu
enable_menuitem	enables a previously-disabled menu item
environment	displays or changes the current dataset and region environment
examine	examines one or more objects, and displays current values (or the values at a specified previous time) in the Main window

Table 2-1. Supported Commands (cont.)

Command name	Action
<code>exit</code>	exits the simulator and the ModelSim application
<code>fcover configure</code>	enables, disables, and sets coverage targets for PSL coverage directives
<code>find</code>	displays the full pathnames of all objects in the design whose names match the name specification you provide
<code>find infiles</code>	searches the specified files and prints to the Transcript window those lines from the files that match the specified pattern.
<code>find insource</code>	searches all source files related to the current design and prints to the Transcript window those lines from the files that match the specified pattern.
<code>formatTime</code>	global format control for all time values displayed in the GUI
<code>force</code>	applies stimulus to VHDL signals and Verilog nets
<code>gdb dir</code>	sets the source directory for FLI/PLI/VPI C source code when using C Debug
<code>getactivecursortime</code>	gets the time of the active cursor in the Wave window
<code>getactivemarkertime</code>	gets the time of the active marker in the List window
<code>help</code>	displays in the Main window a brief description and syntax for the specified command
<code>history</code>	lists the commands executed during the current session
<code>jobspy</code>	controls and monitors batch jobs
<code>layout</code>	allows you to perform operations on GUI layouts
<code>lecho</code>	takes one or more Tcl lists as arguments and pretty-prints them to the Main window
<code>left</code>	searches left (previous) for signal transitions or values in the specified Wave window
<code>log</code>	creates a wave log format (WLF) file containing simulation data for all objects whose names match the provided specifications
<code>lshift</code>	takes a Tcl list as an argument and shifts it in-place one place to the left, eliminating the left-most element
<code>lsublist</code>	returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern
<code>mem compare</code>	compares the selected memory to a reference memory or file
<code>mem display</code>	displays the memory contents of a selected instance to the screen

Table 2-1. Supported Commands (cont.)

Command name	Action
mem list	displays a flattened list of all memory instances in the current or specified context after a design has been elaborated
mem load	updates the simulation memory contents of a specified instance
mem save	saves the contents of a memory instance to a file in any of the supported formats: Verilog binary, Verilog hex, and MTI memory pattern data
mem search	finds and prints to the screen the first occurring match of a specified memory pattern in the specified memory instance
messages clearfilter	removes any filter you have set in the Message Viewer
messages setfilter	performs the same action as the Message Viewer Filter Dialog Box , lets you determine which messages are shown in the Message Viewer
modelsim	starts the ModelSim GUI without prompting you to load a design; valid only for Windows platforms
next	continues a search; see the search command
noforce	removes the effect of any active force commands on the selected object
nolog	suspends writing of data to the WLF file for the specified signals
notepad	opens a simple text editor
noview	closes a window or set of windows in the ModelSim GUI
nowhen	deactivates selected when commands
onbreak	specifies command(s) to be executed when running a macro that encounters a breakpoint in the source code; in effect only during a run command
onElabError	specifies one or more commands to be executed when an error is encountered during elaboration; in effect only during a vsim command
onerror	specifies one or more commands to be executed when a Tcl command in a dofile encounters an error; not dependent on a run command
onfinish	controls simulator behavior when encountering \$finish or sc_stop() in the design code
pause	interrupts the execution of a macro
pop	moves one level up the C callstack
power add	specifies the signals or nets to track for power information

Table 2-1. Supported Commands (cont.)

Command name	Action
power off	works in conjunction with the power add command to make vsim stop updating toggle activity data for the specified signal or net
power on	works in conjunction with the power add command to make vsim begin or resume updating toggle activity data for the specified signal or net
power report	writes out the power information for the specified signals or nets
power reset	resets power information to zero for the signals or nets specified with the power add command
precision	determines how real numbers display in the GUI
printenv	echoes to the Main window the current names and values of all environment variables
process report	creates textual report of all processes displayed in the Process window
profile clear	clears any statistical performance or memory allocation data that has been gathered during previous run commands
profile interval	selects the frequency with which the profiler collects samples during a run command
profile off	disables runtime statistical performance and memory allocation profiling
profile on	enables runtime profiling of where your simulation is spending its time and where memory is allocated
profile option	allows various profiling options to be changed
profile reload	reads in raw profile data from an external file created during memory allocation profiling
profile report	produces a textual output of the profiling statistics that have been gathered up to this point
project	performs common operations on new projects
property list	changes one or more properties of the specified signal, net, or register in the List Window
property wave	changes one or more properties of the specified signal, net, or register in the Wave Window
push	moves one level down the C callstack
pwd	displays the current directory path in the Main window
quietly	turns off transcript echoing for the specified command
quit	exits the simulator

Table 2-1. Supported Commands (cont.)

Command name	Action
qverilog	compiles, optimizes, and simulates a Verilog or SystemVerilog design in one step
radix	specifies the default radix to be used
radix define	creates or modifies a user-defined radix
radix names	returns a list of currently defined radix names
radix list	returns the complete definition of a radix
radix delete	removes the radix definition from the named radix
readers	displays the names of all readers of the specified object
report	displays the value of all simulator control variables, or the value of any simulator state variables relevant to the current simulation
restart	reloads the current dataset if the current dataset is not the active simulation ("sim") and resets the simulation time to zero, in effect acting just like a restart of a simulation
restore	restores the state of a simulation that was saved with a checkpoint command during the current invocation of vsim
resume	continues execution of a macro file after a pause command or a breakpoint
right	searches right (next) for signal transitions or values in the specified Wave window
run	advances the simulation by the specified number of timesteps
runStatus	returns the current state of your simulation after issuing a run or step command
sccom	compiles SystemC design units
scgenmod	creates the equivalent SystemC foreign module declaration for a VHDL entity or Verilog module, and writes it to standard output
sdfcom	compiles SDF files
search	searches the specified window for one or more objects matching the specified pattern(s)
searchlog	searches one or more of the currently open logfiles for a specified condition
see	displays the specified number of source file lines around the current execution line
seetime	scrolls the List or Wave window to make the specified time visible

Table 2-1. Supported Commands (cont.)

Command name	Action
setenv	sets an environment variable
shift	shifts macro parameter values down one place
show	lists objects and subregions visible from the current environment
simstats	reports performance-related statistics about active simulations
status	lists all currently interrupted macros
step	steps to the next HDL statement
stop	stops simulation in batch files; used with the when command
suppress	prevents the specified message(s) from displaying
tb	displays a stack trace for the current process in the Transcript window
tcheck_set	modifies reporting or X generation status of a timing check
tcheck_status	prints the current status of timing checks to the Transcript window
Time	performs various numerical comparisons, operations, and conversions on simulation time values
toggle add	enables collection of toggle statistics for the specified nodes
toggle disable	disables collection of toggle statistics for the specified nodes
toggle enable	re-enables collection of toggle statistics for the specified nodes
toggle report	displays to the Transcript window a list of all nodes that have not transitioned to both 0 and 1 at least once
toggle reset	resets the toggle counts to zero for the specified nodes
tr color	modifies the color of a specific transaction or stream of transactions in a wave window, or all wave windows
tr order	controls which attributes are visible in a transaction and the order in which they appear
tr uid	displays to the Transcript window a list of all active transactions and their IDs
transcribe	displays a command in the Transcript window, then executes the command
transcript	controls echoing of commands executed in a macro file; also works at top level in batch mode
transcript file	sets or queries the pathname for the transcript file

Table 2-1. Supported Commands (cont.)

Command name	Action
<code>tssi2mti</code>	converts a vector file in Technology Standard Events Format (TSSI) into a sequence of <code>force</code> and <code>run</code> commands
<code>typespec</code>	queries class names and class relationships of SystemVerilog classes
<code>ui_VVMode</code>	specifies behavior when encountering user interface registration calls used by verification packages, such as AVM or OVM
<code>unsetenv</code>	deletes an environment variable
<code>up</code>	searches for signal transitions or values in the specified List window
<code>vcd add</code>	adds the specified objects to the VCD file
<code>vcd checkpoint</code>	dumps the current values of all VCD variables to the VCD file
<code>vcd comment</code>	inserts the specified comment in the VCD file
<code>vcd dumpports</code>	creates a VCD file that captures port driver data
<code>vcd dumpportsall</code>	creates a checkpoint in the VCD file that shows the current values of all selected ports
<code>vcd dumpportsflush</code>	flushes the VCD buffer to the VCD file
<code>vcd dumpportslimit</code>	specifies the maximum size of the VCD file
<code>vcd dumpportsoff</code>	turns off VCD dumping and records all dumped port values as x
<code>vcd dumpportson</code>	turns on VCD dumping and records the current values of all selected ports
<code>vcd file</code>	specifies the filename and state mapping for the VCD file created by a <code>vcd add</code> command
<code>vcd files</code>	specifies filenames and state mapping for the VCD files created by the <code>vcd add</code> command; supports multiple VCD files
<code>vcd flush</code>	flushes the contents of the VCD file buffer to the VCD file
<code>vcd limit</code>	specifies the maximum size of the VCD file
<code>vcd off</code>	turns off VCD dumping and records all VCD variable values as x
<code>vcd on</code>	turns on VCD dumping and records the current values of all VCD variables
<code>vcd2wlf</code>	translates VCD files into WLF files
<code>vcom</code>	compiles VHDL design units
<code>vcover attribute</code>	displays attributes in the currently loaded database
<code>vcover merge</code>	merges multiple code or functional coverage data files offline

Table 2-1. Supported Commands (cont.)

Command name	Action
<code>vcover ranktest</code>	ranks the specified input files according to their contribution to cumulative coverage
<code>vcover report</code>	reports on multiple code or functional coverage data files offline
<code>vcover stats</code>	produces summary statistics from multiple coverage data files
<code>vcover testnames</code>	displays test names in the current UCDB file loaded
<code>vdel</code>	deletes a design unit from a specified library
<code>vdir</code>	lists the contents of a design library
<code>vencrypt</code>	encrypts Verilog code contained within encryption envelopes
<code>verror</code>	prints a detailed description of a message number
<code>vgencomp</code>	writes the equivalent VHDL component declaration for a Verilog module to standard output
<code>vhencrypt</code>	encrypts VHDL code contained within encryption envelopes
<code>view</code>	opens a ModelSim window and brings it to the front of the display
<code>virtual count</code>	counts the number of currently defined virtuals that were not read in using a macro file
<code>virtual define</code>	prints the definition of a virtual signal or function in the form of a command that can be used to re-create the object
<code>virtual delete</code>	removes the matching virtuals
<code>virtual describe</code>	prints a complete description of the data type of one or more virtual signals
<code>virtual expand</code>	produces a list of all the non-virtual objects contained in the virtual signal(s)
<code>virtual function</code>	creates a new signal that consists of logical operations on existing signals and simulation time
<code>virtual hide</code>	causes the specified real or virtual signals to not be displayed in the Objects window
<code>virtual log</code>	causes the sim-mode dependent signals of the specified virtual signals to be logged by the simulator
<code>virtual nohide</code>	redispays a virtual previously hidden with virtual hide
<code>virtual nolog</code>	stops the logging of the specified virtual signals
<code>virtual region</code>	creates a new user-defined design hierarchy region
<code>virtual save</code>	saves the definitions of virtuals to a file
<code>virtual show</code>	lists the full path names of all the virtuals explicitly defined

Table 2-1. Supported Commands (cont.)

Command name	Action
virtual signal	creates a new signal that consists of concatenations of signals and subelements
virtual type	creates a new enumerated type
vlib	creates a design library
vlog	compiles Verilog design units and SystemVerilog extensions
vmake	creates a makefile that can be used to reconstruct the specified library
vmap	defines a mapping between a logical library name and a directory
vopt	produces an optimized version of your design
vsim	loads a new design into the simulator
vsim<info>	returns information about the current vsim executable
vsim_break	stop the current simulation before completion
vsource	specifies an alternative file to use for the current source file
wave	commands for manipulating cursors, for zooming, and for adjusting the wave display view in the Wave window
wave create	creates an editable waveform that can be used to create stimulus and drive simulation
wave edit	edits a created waveform
wave export	exports created waveforms to a stimulus file
wave import	imports an EVCD file previously created with a wave export command
wave modify	modifies the parameters of a created waveform
when	instructs ModelSim to perform actions when the specified conditions are met
where	displays information about the system environment
wlf2log	translates a ModelSim WLF file to a QuickSim II logfile
wlf2vcd	translates a ModelSim WLF file to a VCD file
wlfman	outputs information about or a new WLF file from an existing WLF file
wlfrecover	attempts to repair an incomplete WLF file
write cell_report	creates a report of cell instances in the design that are optimized

Table 2-1. Supported Commands (cont.)

Command name	Action
<code>write format</code>	records the names and display options in a file of the objects currently being displayed in the List or Wave window
<code>write list</code>	records the contents of the most recently opened or specified List window in a list output file
<code>write preferences</code>	saves the current GUI preference settings to a Tcl preference file
<code>write report</code>	prints a summary of the design being simulated
<code>write timing</code>	prints timing information about the specified instance
<code>write transcript</code>	writes the contents of the Main window transcript to the specified file
<code>write tssi</code>	records the contents of the default or specified List window in a “TSSI format” file
<code>write wave</code>	records the contents of the most currently opened or specified Wave window in PostScript format
<code>xml2ucdb</code>	converts an XML file into a .ucdb file

.main clear

This command clears the Main window Transcript window.

The behavior is the same as selecting **Edit > Clear** when the Transcript window is active.

Syntax

.main clear

Arguments

None

Related Topics

- [Main Window](#)
- [Transcript Window](#)
- [transcript](#)
- [transcript file](#)

abort

This command halts the execution of a macro file interrupted by a breakpoint or error.

When macros are nested, you may choose to abort the last macro only, abort a specified number of nesting levels, or abort all macros. You can specify this command within a macro to return early.

Syntax

```
abort [<n> | all]
```

Arguments

- `<n>`
(optional) The number of nested macro levels to abort. Specified as an integer greater than 0, where the default value is 1.
- `all`
(optional) Instructs the tool to abort all levels of nested macros.

Related Topics

- [onbreak](#)
- [onElabError](#)
- [onerror](#)

add atv

This command opens an Assertion Thread View (ATV) window for the specified assert or cover directive (designated by its pathname), at the specified evaluation attempt start time.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
add atv <pathname> <time>
```

Arguments

- <pathname>
(required) Specifies the path of an assert or cover directive. Must be specified as the first argument to the **add atv** command.
- <time>
(required) Specifies the start time of an evaluation attempt of the assert or cover directive instance. Must be specified as the second argument to the **add atv** command.

Related Topics

- [atv log](#)
- [Viewing Assertion Threads in the ATV Window](#)

add button

This command adds a user-defined button to the Main window button bar. New buttons are added to the right side of the Standard toolbar.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Returns the path name of the button widget created. You may want to remember this path name, which is similar to:

```
# .dockbar.tbF0.standard.tb.button_49
```

in case you ever want to remove the button.

To remove a button you have previously added you can use the **destroy** Tcl command with the button's path name as an argument, for example:

```
destroy .dockbar.tbF0.standard.tb.button_49
```

Syntax

```
add button <text> {<cmd>[; ...]} [Disable | NoDisable] [{<option> <value> ...}]
```

Arguments

- <text>
(required) Specifies the label to appear on the face of the button. Must be specified as the first argument to the **add button** command.
- <cmd>[; ...]
(required) Specifies the command(s) to be executed when the button is clicked. Multiple commands are specified by separating them with a semicolon (;). Must be specified as the second argument to the **add button** command.

If your command contains any spaces or non-alphanumeric characters you must enclose the command in braces ({ }).

You can echo the command and display the return value in the Transcript window by prefixing the command with the [transcribe](#) command. The **transcribe** command will also echo the results to the Transcript window.
- Disable | NoDisable
(optional) Specifies the appearance of the button.
 - Disable — (default) The button is inactive and grayed-out during a run.
 - NoDisable — The button is active and available during a run.
- <option> <value>
(optional) Specifies Tk button widget(s) that you want to apply to the button. Must be preceded by Disable or NoDisable.
 - <option> — Any Tk button widget.

<value> — Specifies a property of the Tk button widget.

Multiple <option><value> Tk button widgets are entered as a space separated list. You must enclose your option/value pairs in braces ({ }).

Note



To specify any option/value pairs, you must specify either Disable or NoDisable.

You can access the Tk documentation for button widgets by selecting **Help > Tcl Man Pages**, which displays HTML help. You can then select the links: **Tk commands**, then **buttons**.

You can use any properties belonging to Tk button widgets. Useful options are foreground color (**-fg**), background color (**-bg**), width (**-width**), and relief (**-relief**).

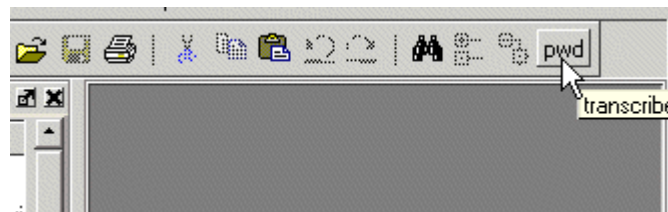
For a complete list of available options, use the configure command addressed to the newly created widget. For example:

```
.dockbar.tbf0.standard.tb.button_51 config
```

Examples

- Create a button labeled “pwd” that invokes the **transcribe** command with the **pwd** Tcl command, and echoes the command and its results to the Transcript window. The button remains active during a run.

```
add button pwd {transcribe pwd} NoDisable
```



- Create a button labeled “date” that echoes the system date to the Transcript window. The button is disabled during a run; its colors are: blue foreground, yellow background, and red active background.

```
add button date {transcribe exec date} Disable \  
{-fg blue -bg yellow -activebackground red}
```

- Create a “doit” button and underline the second character of the label, the “o” of “doit”.

```
add button doit {run 1000 ns; echo did it} Disable {-underline 1}
```

- Change the command that the button executes to “run 10000” and the button’s background color to red; you must know the button’s path name that was returned after the initial creation of the button.

```
.dockbar.tbf0.standard.tb.button_13 config -command {run 10000} -bg red
```

Related Topics

- [transcribe](#)

add dataflow

This command adds the specified process, signal, net, or register to the Dataflow window. Wildcards are allowed.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
add dataflow <object> ... [-connect <source_net> <destination_net>]
    {[-in] [-out] [-inout] | [-ports]} [-internal] [-nofilter] [-recursive]
```

Arguments

- <object> ...
(required unless specifying **-connect**) Specifies a process, signal, net, or register to add to the Dataflow window. Wildcards are allowed. Multiple objects are specified as a space separated list. Refer to the section “[Wildcard Characters](#)” for wildcard usage as it pertains to the add commands. Must be specified as the first argument to the **add dataflow** command.
- -connect <source_net> <destination_net>
(optional) Computes and displays in the Dataflow window all paths between two nets.
 <source_net> — The net that originates the path search.
 <destination_net> — The net that terminates the path search.
- -in
(optional) Specifies to add ports of mode IN.
- -inout
(optional) Specifies to add ports of mode INOUT.
- -out
(optional) Specifies to add ports of mode OUT.
- -ports
(optional) Specifies to add all ports. This switch has the same effect as specifying **-in**, **-out**, and **-inout** together.
- -internal
(optional) Specifies to add internal (non-port) objects.
- -nofilter
(optional) Specifies that the *WildcardFilter* Tcl preference variable be ignored when finding signals or nets.

The *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.

- -recursive

(optional) Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region.

You can specify -r as an alias to this switch.

Examples

- Add all objects in the design to the dataflow window.

add dataflow -r /*

- Add all objects in the region to the dataflow window.

add dataflow *

Related Topics

- [Automatically Tracing All Paths Between Two Nets](#)
- [Dataflow Window](#)
- [Using the WildcardFilter Preference Variable](#)

add list

This command adds the following objects and their values to the List window:

- VHDL signals and variables
- Verilog nets and registers
- User-defined buses
- SystemC primitive channels (signals)

If you do not specify a port mode, such as **-in** or **-out**, this command displays all objects in the selected region with names matching the object name specification.

See “[Wildcard Characters](#)” for wildcard usage as it pertains to the add commands.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
add list {<object> ... | <object_name> {sig ...}} [-allowconstants] [-depth <level>]
  {[<-in>] [<-inout>] [<-out>] | [<-ports>]} [-internal] [-label <name>] [-nodelta] [-optcells]
  [<-<radix_type>> | -radix <type>] [-radixenumnumeric | -radixenumsymbolic] [-recursive]
  [<-trigger> | -nottrigger] [-width <integer>] [-window <wname>]
```

Arguments

- <object> ...
(required when <object_name> {sig ...} is not specified.) Specifies the name of the object to be listed. Multiple objects are entered as a space separated list. Wildcards are allowed. Refer to the section “[Wildcard Characters](#)” for wildcard usage as it pertains to the add commands. Must be specified as the first argument to the **add list** command.

Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.

You can add variables as long as they are preceded by the process name. For example:

```
add list myproc/int1
```

You must specify the <object> argument as the first argument to the **add list** command.

- <object_name> {sig ...}
(required when <object> is not specified) Creates a user-defined bus with the specified object name containing the specified signals (sig) concatenated within the user-defined bus. Arguments, must be enclosed in braces ({ }). Must be specified as the second argument to the **add list** command.

sig — A space-separated list of signals, enclosed in braces ({ }), that are included in the user-defined bus. The signals may be either scalars or various sized arrays as long as they have the same element enumeration type.

For example:

add list {mybus {a b y}}

- -allowconstants

(optional) *For use with wildcard searches.* Specifies that constants matching the wildcard search should be added to the List window.

This command does not add constants by default because they do not change.

- -depth <level>

(optional) Restricts a recursive search, as specified with **-recursive**, to a certain level of hierarchy.

<level> — an integer greater than or equal to zero.

For example, if you specify **-depth 1**, the command descends only one level in the hierarchy.

- -in

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode IN if they match the *object* specification.

- -inout

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode INOUT if they match the *object* specification.

- -out

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode OUT if they match the *object* specification.

- -ports

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include all ports. This switch has the same effect as specifying **-in**, **-out**, and **-inout** together.

- -internal

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include internal objects (non-port objects) if they match the *object* specification. VHDL variables are not selected.

- -label <name>

(optional) Specifies an alternative signal name to be displayed as a column heading in the listing.

<name> — Specifies the label to be used at the top of the column. You must enclose <name> in braces ({ }) if it includes any spaces.

This alternative name is not valid in a [force](#) or [examine](#) command. However, you can use the alternate name when invoking the [search](#) command with <window_name> specified as **list**.

- **-nodelta**
(optional) Specifies that the delta column not be displayed when adding signals to the List window. Identical to [configure list -delta none](#).
- **-optcells**
(optional) *For use with wildcard searches.* Allows Verilog optimized cell ports to be visible when using wildcards. By default Verilog optimized cell ports are not selected even if they match the specified wildcard pattern.
- **-<radix_type>**
(optional) Specifies the radix type for the objects that follow in the command. Valid entries (or any unique abbreviations) are: binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default.

If no radix is specified for an enumerated type, the default radix is used. You can change the default radix for the current simulation using the [radix](#) command. You can change the default radix permanently by editing the [DefaultRadix](#) variable in the *modelsim.ini* file.

If you specify a radix for an array of a VHDL enumerated type, ModelSim converts each signal value to 1, 0, Z, or X.
- **-radix <type>**
(optional) Specifies a user-defined radix. The **-radix <type>** switch can be used in place of the **-<radix_type>** switch. For example, **-radix hexadecimal** is the same as **-hex**.

<type> — binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default.

This option overrides the global setting of the default radix (the [DefaultRadix](#) variable in the *modelsim.ini* file) for the current simulation only.
- **-radixenumnumeric**
This option overrides the global setting of the default radix (the [DefaultRadix](#) variable in the *modelsim.ini* file).
- **-radixenumsymbolic**
(optional) Reverses the action of **-radixenumnumeric** and sets the global setting of the default radix (the [DefaultRadix](#) variable in the *modelsim.ini* file) to symbolic.
- **-recursive**
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region. You can use the **-depth** argument to specify how far down the hierarchy to descend. You can use **"-r"** as an alias to this switch.
- **-trigger** | **-nottrigger**
(optional) Specifies whether objects should be updated in the List window when the objects change value.

-trigger — (default) Update objects in the List Window when their values change.

-notrigger — Do not update objects in the List Window when their values change.

- **-width <integer>**

(optional) Formats the column width.

integer — A positive integer specifying the column width in characters.

- **-window <wname>**

(optional) Adds objects to the specified List window (e.g., list2).

<wname> — The window to add objects to. Used to specify a particular window when multiple instances of that window type exist.

This option selects an existing window, but does not create a new window. Use the [view](#) command with the **-new** option to create a new window.

Examples

- List all objects in the design.

```
add list -r /*
```

- List all objects in the region.

```
add list *
```

- List all input ports in the region.

```
add list -in *
```

- Display a List window containing three columns headed *a*, *sig*, and *array_sig(9 to 23)*.

```
add list a -label sig /top/lower/sig {array_sig(9 to 23)}
```

- List *clk*, *a*, *b*, *c*, and *d* only when *clk* changes.

```
add list clk -notrigger a b c d
```

- Lists *clk*, *a*, *b*, *c*, and *d* every 100 ns.

```
config list -strobeperiod {100 ns} -strobestart {0 ns} -usestrobe 1  
add list -notrigger clk a b c d
```

- Creates a user-defined bus named "mybus" consisting of three signals; the bus is displayed in hex.

```
add list -hex {mybus {msb {opcode(8 downto 1)} data}}
```

- Lists the object *vec1* using symbolic values, lists *vec2* in hexadecimal, and lists *vec3* and *vec4* in decimal.

```
add list vec1 -hex vec2 -dec vec3 vec4
```

- Open a new List window with "SV_Signals" as its title, then add signals to it.

```
set SV_Signals [view list -new -title SV_Signals]
```

```
add list -window $SV_Signals /top/mysignals
```

The custom window title "SV_Signals" is saved as a TCL variable, then called using the '\$' prefix.

Related Topics

- [add wave](#)
- [log](#)
- [Extended Identifiers](#)
- [Using the WildcardFilter Preference Variable](#)

add memory

This command displays the contents and sets the address and data radix of the specified memory in the MDI frame of the Main window.

See “[Wildcard Characters](#)” for wildcard usage as it pertains to the add commands.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
add memory [-addressradix {decimal | hex}] [-dataradix <type>]  
          [-radixenumnumeric | -radixenumsymbolic] [-wordspersline <num>] <object_name> ...
```

Arguments

- -addressradix {[decimal](#) | hex}
(optional) Specifies the address radix for the memory display.
 - decimal — (default) Sets the radix to decimal. You can abbreviate this argument to "d".
 - hex — Sets the radix to hexadecimal. You can abbreviate this argument to "h".
- -dataradix <type>
(optional) Specifies the data radix for the memory display. If you do not specify this switch, the command uses the global default radix.
 - <type> — binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default.If you do not specify a radix for an enumerated type, the command uses the symbolic representation.

You can change the default radix for the current simulation using the [radix](#) command. You can change the default radix permanently by editing the [DefaultRadix](#) variable in the *modelsim.ini* file. Changing the default radix does not change the radix of the currently displayed memory. Use the **add memory** command to re-add the memory with the desired radix, or change the display radix from the Memory window Properties dialog.
- -radixenumnumeric
(optional) Causes Verilog and SystemC enums to be displayed as numbers (formatted by the current radix). This overrides the default behavior of always showing enums symbolically.
- -radixenumsymbolic
(optional) Restores the default behavior of displaying Verilog and SystemC enums as symbols by reversing the action of the **-radixenumnumeric** option.
- -wordspersline <num>
(optional) Specifies how many words are displayed on each line in the memory window. By default, the information displayed will wrap depending on the width of the window.
 - num — Any positive integer

- <object_name> ...

(required) Specifies the hierarchical path of the memory to be displayed. Multiple memories are specified as a space separated list. Must be specified as the final argument to the **add memory** command.

Wildcard characters are allowed.

Note

The *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.)

Related Topics

- [Memory List Window](#)
- [Using the WildcardFilter Preference Variable](#)

add schematic

This command adds the specified process, signal, net, or register to the Schematic window. Wildcards are allowed.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
add schematic <object> ... [-connect <source_net> <destination_net>]
    {[-in] [-out] [-inout] | [-ports]} [-internal] [-nofilter] [-recursive] [-window <wname>]
```

Arguments

- <object> ...
(required unless specifying **-connect**) Specifies a process, signal, net, or register to add to the Schematic window. Wildcards are allowed. Multiple objects are specified as a space separated list, Refer to the section “[Wildcard Characters](#)” for wildcard usage as it pertains to the add commands. Must be specified as the first argument to the **add schematic** command.
- -connect <source_net> <destination_net>
(optional) Computes and displays in the Schematic window all paths between two nets.
 <source_net> — The net that originates the path search.
 <destination_net> — The net that terminates the path search.
- -in
(optional) Specifies to add ports of mode IN.
- -inout
(optional) Specifies to add ports of mode INOUT.
- -out
(optional) Specifies to add ports of mode OUT.
- -ports
(optional) Specifies to add all ports. This switch has the same effect as specifying **-in**, **-out**, and **-inout** together.
- -internal
(optional) Specifies to add internal (non-port) objects.
- -nofilter
(optional) Specifies that the *WildcardFilter* Tcl preference variable be ignored when finding signals or nets.

The *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.

- **-recursive**
(optional) Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region.
You can specify **-r** as an alias to this switch.
- **-window <wname>**
(optional) Adds the object(s) to the specified Schematic window.
<wname> — the name of the schematic window, as shown in the window's tab.
This switch is useful when you have multiple schematic windows open.
You can open a new schematic window by entering:
view schematic -new

Examples

- Add all objects in the design to the schematic window.
add schematic -r /*
- Add all objects in the region to the schematic window.
add schematic *
- Open a new Schematic window with "SFLOW" as its title, then add signals to it.
set SFLOW [view schematic -new -title SFLOW]
add schematic -window \$SFLOW /top/mysignals

The custom window title "SFLOW" is saved as a TCL variable, then called using the '\$' prefix.

Related Topics

- [Schematic Window](#)
- [Using the WildcardFilter Preference Variable](#)

add testbrowser

This command adds *.ucdb* file(s) to the test management browser.

Syntax

```
add testbrowser <ucdb_filename> [<ucdb_filename>...]
```

Arguments

- <ucdb_filename> [<ucdb_filename>...]
(required: at least one *.ucdb*) Specifies the name of the *.ucdb* file(s) to be added. Multiple filenames are specified as a space separated list.

Wildcard characters are allowed. (Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.)

Related Topics

- [“Verification Browser Window”](#)
- [Using the WildcardFilter Preference Variable](#)

add watch

This command adds signals and variables to the Watch window in the Main window. SystemC objects and user-defined buses may also be added.

See “[Wildcard Characters](#)” for wildcard usage as it pertains to the add commands.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
add watch <object_name> ... [-radix <type>] [-radixenumnumeric | -radixenumsymbolic]
```

Arguments

- <object_name> ...
(required) Specifies the name of the object to be added. Multiple objects are entered as a space separated list. Must be specified as the first argument to the **add watch** command.
Wildcard characters are allowed. (Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.)
Variables must be preceded by the process name. For example,

```
add watch myproc/int1
```
- -radix <type>
(optional) Specifies a user-defined radix. If you do not specify this switch, the command uses the global default radix.

<type> — binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default.
You can change the default radix for the current simulation using the [radix](#) command. You can change the default radix permanently by editing the [DefaultRadix](#) variable in the *modelsim.ini* file.
- -radixenumnumeric
(optional) Causes Verilog and SystemC enums to be displayed as numbers (formatted by the current radix). This overrides the default behavior of always showing enums symbolically.
- -radixenumsymbolic
(optional) Restores the default behavior of displaying Verilog and SystemC enums as symbols by reversing the action of the **-radixenumnumeric** option.

Related Topics

- [Watch window](#)
- [Using the WildcardFilter Preference Variable](#)

add wave

This command adds the following objects to the Wave window:

- VHDL signals and variables
- Verilog nets and registers
- SystemVerilog class objects
- SystemC primitive channels (signals)
- Dividers and user-defined buses.

If no port mode is specified, this command will display all objects in the selected region with names matching the object name specification.

See “[Wildcard Characters](#)” for wildcard usage as it pertains to the add commands.

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

```
add wave [-allowconstants] [-clampanalog {0 | 1}] [-color <standard_color_name>]
[-depth <level>] [[-divider [<divider_name> ...] [-expand <signal_name>]
[-format <type> | -<format>] [-group <group_name> [<sig_name1> ...]] [-height <pixels>]
{[-in] [-inout] [-out] | [-ports]] [-internal] [-label <name>] [-max <real_num>]
[-min <real_num>] [-mvcall] [-mvcovm] [-noupdate] [-optcells] [-position <location>]
[-<radix_type> | -radix <type>] [-radixenumnumeric | -radixenumsymbolic] [-recursive]
[-time] [-window <wname>] [<object_name> ...] [{<object_name> {sig1 sig2 ...}}]
```

Arguments

- -allowconstants
(optional) *For use with wildcard searches.* Specifies that constants matching the wildcard search should be added to the Wave window.
By default, constants are ignored because they do not change.
- -clampanalog {0 | 1}
(optional) Clamps the display of an analog waveform to the values specified by **-max** and **-min**. Specifying a value of 1 prevents the waveform from extending above the value specified for **-max** or below the value specified for **-min**.
 - 0 — not clamped
 - 1 — (default) clamped
- -color <standard_color_name>
(optional) Specifies the color used to display a waveform.
<standard_color_name> — You can use either of the following:

standard X Window color name — enclose 2-word names in quotes ("), for example:

-color "light blue"

rgb value — for example:

-color #357f77

- **-depth <level>**

(optional) Restricts a recursive search, as specified with **-recursive** to a specified level of hierarchy.

<level> — Any integer greater than or equal to zero. For example, if you specify **-depth 1**, the command descends only one level in the hierarchy.

- **-divider [<divider_name> ...]**

(optional) Adds a divider to the Wave window. If you do not specify this argument, the command inserts an unnamed divider.

<divider_name> ... — Specifies the name of the divider, which appears in the pathnames column. Multiple objects entered as a space separated list.

When you specify more than one **<divider_name>** the command creates a divider for each name.

You can begin a name with a space, but you must enclose the name within quotes (") or braces ({ }) You cannot begin a name with a hyphen (-).

- **-expand <signal_name>**

(optional) Instructs the command to expand a compound signal immediately, but only one level down.

<signal_name> — Specifies the name of the signal. This string can include wildcards.

- **-format <type> | -<format>**

(optional) Specifies the display format of the objects. Valid entries are:

-format <type>	-<format>	Display Format
-format literal	-literal	Literal waveforms are displayed as a box containing the object value.
-format logic	-logic	Logic signals may be U, X, 0, 1, Z, W, L, H, or '-'.
-format analog-step	-analog-step	Analog-step changes to the new time before plotting the new Y.
-format analog-interpolated	-analog-interpolated	Analog-interpolated draws a diagonal line.
-format analog-backstep	-analog-backstep	Analog-backstep plots the new Y before moving to the new time.

-format event	-event	Displays a mark at every transition.
---------------	--------	--------------------------------------

The way each state is displayed is specified by the logic type display preference (refer to [modelsim.ini Variables](#)).

The Y-axis range of analog signals is bounded by **-max** and **-min** switches.

- -group <group_name> [<sig_name1> ...]

(optional) Creates a wave group with the specified group_name.

<group_name> — Specifies the name of the group. You must enclose this argument in quotes (") or braces ({ }) if it contains any spaces.

<sig_name> ... — Specifies the signals to add to the group. Multiple signals are entered as a space separated list. This command creates an empty group if you do not specify any signal names.

- -height <pixels>

(optional) Specifies the height of the waveform in pixels.

<pixels> — Any positive integer.

- -in

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode IN if they match the object_name specification.

- -out

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode OUT if they match the object_name specification.

- -inout

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode INOUT if they match the object_name specification.

- -ports

(optional) *For use with wildcard searches.* Specifies that the scope of the listing is to include ports of modes IN, OUT, or INOUT.

- -internal

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include internal objects (non-port objects) if they match the object_name specification.

- -label <name>

(optional) Specifies an alternative name for the signal being added. For example,

add wave -label c clock

adds the *clock* signal, labeled as "c".

This alternative name is not valid in a [force](#) or [examine](#) command; however, it can be used in a [search](#) command with the **wave** option.

- **-max** <real_num>
(optional) Specifies the maximum Y-axis data value to be displayed for an analog waveform. Used in conjunction with the **-min** switch; the value you specify for **-max** must be greater than the value you specify for **-min**.
<real_num> — Any integer that is greater than the value specified for **-min**.
- **-min** <real_num>
(optional) Specifies the minimum Y-axis data value to be displayed for an analog waveform. Used in conjunction with the **-max** switch; the value you specify for **-min** must be less than the value you specify for **-max**.
<real_num> — Any integer that is less than the value specified for **-max**.

For example, if you know the Y-axis data for a waveform varies between 0.0 and 5.0, you could add the waveform with the following command:

```
add wave -analog -min 0 -max 5 -height 100 my_signal
```

Note



Although **-offset** and **-scale** are still supported, the **-max** and **-min** arguments provide an easier way to define upper and lower limits of an analog waveform.

- **-mvcall**
(optional) Specifies the inclusion of all MVC protocol transactions when a wildcard is used (i.e. add wave -r /*). By default, MVC transactions are sometimes excluded from view with normal wildcard usage.
- **-mvcovm**
(optional) Specifies the inclusion of all OVM sequence transactions when a wildcard is used (i.e. add wave -r /*). By default, OVM sequence transactions are sometimes excluded from view with normal wildcard usage.
- **-noupdate**
(optional) Prevents the Wave window from updating when a series of add wave commands are executed in series.
- **-optcells**
(optional) Specifies that optimized cell ports are visible when using wildcards. By default optimized cell ports are not selected even if they match the specified wildcard pattern.
- **-position** <location>
(optional) Specifies where the command adds the signals.
<location> — Can be any of the following:
 - top — Adds the signals to the beginning of the list of signals.
 - bottom | end — Adds the signals to the end of the list of signals.

before | above — Adds the signals to the location before the first selected signal in the wave window.

after | below — Adds the signals to the location after the first selected signal in the wave window.

<integer> — Adds the signals beginning at the specified point in the list of signals.

- -<radix_type>

(optional) Specifies the radix type for the objects that follow in the command. Valid entries (or any unique abbreviations) are: binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default.

If no radix is specified for an enumerated type, the default radix is used. You can change the default radix for the current simulation using the [radix](#) command. You can change the default radix permanently by editing the [DefaultRadix](#) variable in the *modelsim.ini* file.

If you specify a radix for an array of a VHDL enumerated type, ModelSim converts each signal value to 1, 0, Z, or X.

- -radix <type>

(optional) Specifies a user-defined radix. The -radix <type> switch can be used in place of the -<radix_type> switch. For example, -radix hexadecimal is the same as -hex.

<type> — binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default.

This option overrides the global setting of the default radix (the [DefaultRadix](#) variable in the *modelsim.ini* file) for the current simulation only.

- -radixenumnumeric

(optional) Causes Verilog and SystemC enums to be displayed as numbers (formatted by the current radix). This overrides the default behavior of always showing enums symbolically.

- -radixenumsymbolic

(optional) Restores the default behavior of displaying Verilog and SystemC enums as symbols by reversing the action of the **-radixenumnumeric** option.

- -recursive

(optional) *For use with wildcard searches.* Specifies that the scope of the search is to descend recursively into subregions.

If you do not specify this switch, the search is limited to the selected region. You can use the **-depth** argument to specify how far down the hierarchy to descend.

- -time

(optional) Use time as the radix for Verilog objects that are register-based types (register vectors, time, int, and integer types).

- -window <wname>

(optional) Adds objects to a specified window. Used to specify a particular wave window when multiple wave windows exist.

<wname> — The name of the wave window (for instance, wave2). Must be an existing window. Does not create a new window.

Use the **view** command with the **-new** option to create a new window.

- <object_name> ...

(required unless specifying {<object_name> {sig1 sig2 ...}}) Specifies the names of objects to be included in the Wave window. Must be specified as the final argument to the **add wave** command. Wildcard characters are allowed. Multiple objects are entered as a space separated list. Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.

Variables may be added if preceded by the process name. For example,

```
add wave myproc/int1
```

- {<object_name> {sig1 sig2 ...}}

(required unless specifying <object_name>) Creates a user-defined bus with the specified object name containing the specified signals (sig1 and so forth) concatenated within the user-defined bus. Must be specified as the final argument to the **add wave** command.

sig — A space-separated list of signals, enclosed in braces ({ }), that are included in the user-defined bus. The signals may be either scalars or various sized arrays as long as they have the same element enumeration type.

Note

You can also select **Wave > Combine Signals** (when the Wave window is selected) to create a user-defined bus.

Examples

- Display an object named *out2*. The object is specified as being a logic object presented in gold.

```
add wave -logic -color gold out2
```

- Display a user-defined, hex formatted bus named *address*.

```
add wave -hex {address {a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0}}
```

- Add all wave objects in the region.

```
add wave *
```

- Add all wave input ports in the region.

```
add wave -in *
```

- Create a user-defined bus named "mybus" consisting of three signals. *Scalar1* and *scalar2* are of type `std_logic` and *vector1* is of type `std_logic_vector` (7 downto 1). The bus is displayed in hex.

```
add wave -hex {mybus {scalar1 vector1 scalar2}}
```

Slices and arrays may be added to the bus using either VHDL or Verilog syntax. For example:

```
add wave {vector3(1)}
add wave {vector3[1]}
add wave {vector3(4 downto 0)}
add wave {vector3[4:0]}
```

- Add the object *vec1* to the Wave window using symbolic values, adds *vec2* in hexadecimal, and adds *vec3* and *vec4* in decimal.

```
add wave vec1 -hex vec2 -dec vec3 vec4
```

- Add a divider with the name "-Example-". Note that for this to work, the first hyphen of the name must be preceded by a space.

```
add wave -divider " -Example- "
```

- Add an unnamed divider.

```
add wave -divider
add wave -divider ""
add wave -divider {}
```

- Open a new Wave window with "SV_Signals" as its title, then add signals to it.

```
set SV_Signals [view wave -new -title SV_Signals]
add wave -window $SV_Signals /top/mysignals
```

The custom window title "SV_Signals" is saved as a TCL variable, then called using the '\$' prefix.

Related Topics

- [add list](#)
- [log](#)
- [Concatenation Directives](#)
- [Extended Identifiers](#)
- [Using the WildcardFilter Preference Variable](#)
- See [Wave Window](#) for more information on analog formats of waveform signals.

add_cmdhelp

This command adds the specified command name, description, and command arguments to the command-line help. You can then access the information using the [help](#) command.

To delete an entry, invoke the command with an empty command description and arguments. See examples.

The arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
add_cmdhelp {<command_name>} {<command_description>} {<command_arguments>}
```

Arguments

- {<command_name>}
(required) Specifies the command name that will be entered as an argument to the **help** command. Must be enclosed in braces ({ }). The command_name must not interfere with an already existing command_name. Must be specified as the first argument to the **add_cmdhelp** command.
- {<command_description>}
(required) Specifies a description of the command. Must be enclosed in braces ({ }). Must be specified as the second argument to the **add_cmdhelp** command.
- {<command_arguments>}
(required) A space-separated list of arguments for the command. Must be enclosed in braces ({ }). If the command doesn't have any arguments, enter {}. Must be specified as the third argument to the **add_cmdhelp** command.

Examples

- Add a command named "date" with no arguments.

```
add_cmdhelp date {Displays date and time.} {}
```

Entering:

```
VSIM> help date
```

returns:

```
Displays date and time.  
Usage: date
```

- Add the change date command.

```
add_cmdhelp {change date} {Modify date or time.} {-time|-date <arg>}
```

Entering:

```
VSIM> help change date
```

returns:

Modify data or time.

Usage: change date -time|-date <arg>

- Deletes the change date command from the command-line help.

add_cmdhelp {change date} {} {}

add_menu

This command adds a menu to the menu bar of the specified window, using the specified menu name. Use the [add_menuitem](#), [add_separator](#), [add_menubc](#), and [add_submenu](#) commands to complete the menu.

Returns the full Tk pathname of the new menu.

Color and other Tk properties of the menu may be changed, after creating the menu, using the Tk menu widget configure command.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
add_menu <window_name> <menu_name> [<shortcut> [-hide_menubutton]]
```

Arguments

- <window_name>
(required) Specifies the Tk path of the window to contain the menu. Must be specified as the first argument to the **add_menu** command.

To add a menu to the Main window you must express this value as: `""`. For example,

```
add_menu "" mymenu
```

To add a menu to any other window, you need to determine the `window_name` by entering the view command. For example:

```
view wave
```

returns:

```
# .main_pane.wave
```

Note that the `<window_name>` for each window, except the Main window, begins with a period (`.`).

- <menu_name>
(required) Specifies the name to be given to the Tk menu widget. Must be specified as the first argument to the **add_menu** command.
- <shortcut>
(optional) An integer that specifies the number of the letter in the menu name that is to be used as the shortcut where the default is -1, no shortcut. Numbering starts with 0 (first letter = 0, second letter = 1, third letter = 2, and so on). Optional unless you specify **-hide_menubutton**, in which case `<shortcut>` is required. Default is `"-1"`, indicating no shortcut is to be used.

- `-hide_menubutton`

(optional) Specifies that the new menu is not to be displayed. You can add the menu later by calling `tk_popup` on the menu path widget. Note that you must specify `<shortcut>` if you specify `-hide_menubutton`.

Examples

The following Tcl code is an example of creating user-customized menus. It adds a menu containing a top-level item labeled "Do My Own Thing...", which prints "my_own_thing.signals", and adds a cascading submenu labeled "changeCase" with two entries, "To Upper" and "To Lower", which echo "my_to_upper" and "my_to_lower" respectively. A checkbox that controls the value of myglobalvar (.signals:one) is also added.

```
set myglobalvar 0
set wname [view wave]; #Gets path to Wave window
proc AddMyMenus {wname} {
    global myglobalvar
    set cmd1 "echo my_own_thing $wname"
    set cmd2 "echo my_to_upper $wname"
    set cmd3 "echo my_to_lower $wname"

#           WindowName  Menu           MenuItem label      Command
#           -----
add_menu    $wname      mine
add_menuitem $wname      mine           "Do My Own Thing..." $cmd1
add_separator $wname      mine           ;#-----
add_submenu $wname      mine           changeCase
add_menuitem $wname      mine.changeCase "To Upper"         $cmd2
add_menuitem $wname      mine.changeCase "To Lower"         $cmd3
add_submenu $wname      mine           vars
add_menuchb $wname      mine.vars       "Feature One"      -variable
                                     myglobalvar
                                     -onvalue 1
                                     -offvalue 0
                                     -indicatoron 1
}
AddMyMenus $wname
```

This example is available in the following DO file:

```
<install_dir>/examples/misc/addmenu.do.
```

You can run the DO file to add the "Mine" menu shown in the illustration, or modify the file for different results.

To execute the DO file, select **Tools > Execute Macro** (Main window), or use the `do` command.

Related Topics

- [add_menuchb](#)
- [add_menuitem](#)
- [add_separator](#)
- [add_submenu](#)
- [change_menu_cmd](#)

add_menucb

This command creates a checkbox within the specified menu of the specified window. A checkbox is a small box with a label. Clicking on the box will toggle the state, from on to off or the reverse.

When the box is "on", the Tcl global variable <var> is set to <onval>. When the box is "off", the global variable is set to <offval>. Also, if something else changes the global variable, its current state is reflected in the state of the checkbox. Returns nothing.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
add_menucb <window_name> <menu_name> <text> [-indicatoron {0 | 1}]  
          [-onvalue <onval>] [-offvalue <offval>] [-variable <var>]
```

Arguments

- <window_name>
(required) Specifies the Tk path of the window to contain the menu. Must be specified as the first argument to the **add_menucb** command.

To add a menu to the Main window you must express this value as: `""`. For example:

```
add_menucb "" mymenu
```

To add a menu to any other window, you must determine the `window_name` by executing the **view** command, for example:

```
view wave
```

returns:

```
# .main_pane.wave
```

Note that all windows, other than the Main window, begin with a period (.).

- <menu_name>
(required) Specifies the name of the Tk menu widget. Must be specified as the second argument to the **add_menucb** command.
- <text>
(required) Specifies the text to be displayed next to the checkbox. Must be specified as the third argument to the **add_menucb** command.
- -indicatoron {0 | 1}
(optional) Specifies whether or not the status indicator is displayed.

0 — off

1 — (default) on

- **-onvalue <onval>**
(optional) Specifies the value of the global Tcl variable when the box is "on".
 <onval> — A value specific to the global Tcl variable specified in -variable.
- **-offvalue <offval>**
(optional) Specifies the value of the global Tcl variable when the box is "off".
 <offval> — A value specific to the global Tcl variable specified in -variable.
- **-variable <var>**
(optional) Specifies the global Tcl variable to be reflected and changed.
 <var> — A global Tcl variable.

Examples

```
add_menucb $wname mine.vars "Feature One" \  
-variable myglobalvar($wname:one) -onvalue 1 -offvalue 0 -indicatoron 1
```

This command is also used as part of the [add_menu](#) example.

Related Topics

- [add_menu](#)
- [add_menuitem](#)
- [add_separator](#)
- [add_submenu](#)
- [change_menu_cmd](#)

add_menuitem

This command creates a menu item within the specified menu of the specified window. May be used within a submenu.

Returns nothing.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
add_menuitem <window_name> <menu_path> <text> <cmd> [<shortcut>]
```

Arguments

The arguments <window_name>, <menu_path>, <text>, and <cmd> are order dependent and must be entered as specified in the Syntax section.

- <window_name>

(required) Specifies the Tk path of the window to contain the menu. Must be specified as the first argument to the **add_menuitem** command.

To add a menu to the Main window you must express this value as: "". For example,

```
add_menu "" mymenu
```

To add a menu to any other window, you must determine the window_name by executing the **view** command, for example:

```
view wave
```

returns:

```
# .main_pane.wave
```

Note that all windows, other than the Main window, begin with a period (.).

- <menu_path>

(required) Specifies the name of the Tk menu widget plus submenu path. Must be specified as the second argument to the **add_menuitem** command.

- <text>

(required) Specifies the text to be displayed. Must be specified as the third argument to the **add_menuitem** command.

- <cmd>

(required) Specifies the command to be executed when the menu item is selected with the left mouse button. Must be specified as the fourth argument to the **add_menuitem** command.

To echo the command and display the return value in the Main window, prefix the command with the [transcribe](#) command. The **transcribe** command will also echo the results to the Transcript window.

- <shortcut>
(optional) An integer that specifies the number of the letter in the menu name that is to be used as the shortcut where the default is -1, no shortcut. Numbering starts with 0 (first letter = 0, second letter = 1, third letter = 2, and so on).

Examples

```
add_menuitem $wname user "Save Results As..." $my_save_cmd
```

This command is also used as part of the [add_menu](#) example.

Related Topics

- [add_menu](#)
- [add_menuchb](#)
- [add_separator](#)
- [add_submenu](#)
- [change_menu_cmd](#)

add_separator

This command adds a separator as the next item in the specified menu path in the specified window.

Returns nothing.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
add_separator <window_name> <menu_path>
```

Arguments

The arguments <**window_name**> and <**menu_path**> are order dependent and must be entered as specified in the Syntax section.

- <window_name>

(required) Specifies the Tk path of the window to contain the menu. Must be specified as the first argument to the **add_separator** command.

To add a menu to the Main window you must express this value as: `""`. For example,

```
add_menu "" mymenu
```

To add a menu to any other window, you must determine the window_name by executing the **view** command, for example:

```
view wave
```

returns:

```
# .main_pane.wave
```

Note that all windows, other than the Main window, begin with a period (.).

- <menu_path>

(required) Specifies the name of the Tk menu widget plus submenu path. Must be specified as the second argument to the **add_separator** command.

Examples

```
add_separator $wname user
```

This command is also used as part of the [add_menu](#) example.

Related Topics

- [add_menu](#)
- [add_menub](#)
- [add_menuitem](#)
- [add_submenu](#)
- [change_menu_cmd](#)

add_submenu

This command creates a cascading submenu within the specified menu path of the specified window. May be used within a submenu.

Returns the full Tk path to the new submenu widget.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
add_submenu <window_name> <menu_path> <name> [<shortcut>]
```

Arguments

The arguments <window_name>, <menu_path>, and <name> are order dependent and must be specified as shown in the Syntax section.

- <window_name>
(required) Specifies the Tk path of the window to contain the menu. Must be specified as the first argument to the **add_submenu** command.

To add a menu to the Main window you must express this value as: `""`. For example,

```
add_menu "" mymenu
```

To add a menu to any other window, you must determine the window_name by executing the view command, for example:

```
view wave
```

returns:

```
# .main_pane.wave
```

Note that all windows, other than the Main window, begin with a period (.).

- <menu_path>
(required) Specifies the name of the Tk menu widget plus submenu path. Must be specified as the second argument to the **add_submenu** command.
- <name>
(required) Specifies the name to be displayed on the submenu. Must be specified as the third argument to the **add_submenu** command.
- <shortcut>
(optional) An integer that specifies the number of the letter in the menu name that is to be used as the shortcut where the default is -1, no shortcut. Numbering starts with 0 (first letter = 0, second letter = 1, third letter = 2, and so on).

Examples

This command is used as part of the [add_menu](#) example.

Related Topics

- [add_menu](#)
- [add_menuch](#)
- [add_menuitem](#)
- [add_separator](#)
- [change_menu_cmd](#)

alias

This command displays or creates user-defined aliases. Any arguments passed on invocation of the alias will be passed through to the specified commands.

Returns nothing. Existing commands (e.g., **run**, **env**, etc.) cannot be aliased.

Syntax

```
alias [<name> ["<cmds>"]]
```

Arguments

- <name>
(optional) Specifies the new procedure name to be used when invoking the commands.
- "<cmds>"
(optional) Specifies the command or commands to be evaluated when the alias is invoked. Multiple commands are specified as a semicolon (;) separated list. You must enclose the string in quotes ("").

Examples

- List all aliases currently defined.

```
alias
```
- List the alias definition for the specified name if one exists.

```
alias <name>
```
- Create a Tcl procedure, "myquit", that when executed, writes the contents of the List window to the file *mylist.save* by invoking [write list](#), and quits ModelSim by invoking [quit](#).

```
alias myquit "write list ./mylist.save; quit -f"
```

assertion action

This command allows you to set the assertion action for concurrent assertion starts, failures, passes, and antecedent matches. The actions can be: continue, break, exit, or a tcl subroutine call. The action setting for start, antecedent and pass conditions works only if:

- the assertion is browseable – i.e., the **+acc=a** argument is used with the **vopt** command;
- and Assertion debug is set – i.e., the **-assertdebug** switch is used with the **vsim** command).

You can define a tcl subroutine in a *.do* file and call it on a specific assertion event.

Syntax

```
assertion action -cond [start | antecedent | pass | fail] -exec [continue | break | exit |
    tcl_subroutine] [-assert | -cover] [-recursive] <path> [<path> ...]
```

Arguments

- -cond [start | antecedent | pass | fail]
(required) Designates the assertion condition as start, antecedent, pass or fail.
- -exec [continue | break | exit | tcl_subroutine]
(required) Specifies the type of action to take on the designated assertion condition.
 - continue — (default) No action taken. This is the default value if you do not specify this switch.
 - break — Halt simulation and return to the ModelSim prompt.
 - exit — Halt simulation and exit ModelSim.
 - tcl_subroutine — Execute tcl subroutine.
- -assert | -cover
(optional) The -assert switch specifies that the action applies only to assertions. The -cover switch specifies that the action applies only to cover directives. If neither is specified, the action applies to both assertions and cover directives.
- -recursive
(optional) For use with wildcard matching. Specifies that the scope of the matching is to descend recursively into subregions. If omitted, the search is limited to the selected region. Applies to all paths specified in the command. You can abbreviate this switch to “-r”.
- <path> [<path> ...]
(required) A space separated list of paths, that specifies the assertions to be affected. Multiple paths and wildcards are allowed. The path specifies assertions or a design region containing multiple assertions. Must be specified as the final argument to the **assertion active** command.

You must specify at least one <path> argument, but can also specify more in a space separated list.

Related Topics

- [assertion active](#)
- [assertion count](#)
- [assertion enable](#)
- [assertion fail](#)
- [assertion pass](#)
- [assertion profile](#)
- [atv log](#)
- [Viewing Assertions in the Assertions Window](#)

assertion active

This command instructs the simulator to report on any active assertion directives at the end of simulation (EOS). Active assertion directives will be indicated in the Assertions tab of the Analysis window with the text "active at end of simulation" in the EOS Note column. If the PSL assert directive is strong, the EOS Note column will report both the usual "active at end of simulation" note along with a PSL strong error message.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
assertion active [-eosnote {off | on}] [-recursive] <path> [<path> ...]
```

Arguments

- -eosnote {off | on}
(optional) Controls the reporting of active assertion directives in the EOS Note column of the Assertions tab of the Analysis window. The EOS Note message scopes will match those that appear in assertion messages during runtime.
 - off — (default) Reporting of active assertion directives turned off.
 - on — Reporting of active assertion directives turned on.
- -recursive
(optional) *For use with wildcard matching.* Specifies that the scope of the matching is to descend recursively into subregions. If omitted, the search is limited to the selected region. Applies to all paths specified in the command. You can abbreviate this switch to “-r”.
- <path> [<path> ...]
(required) A space separated list of paths, that specifies the assertions to be affected. Multiple paths and wildcards are allowed. The path specifies assertions or a design region containing multiple assertions. Must be specified as the final argument to the **assertion active** command.
You must specify at least one <path> argument, but can also specify more in a space separated list.

Examples

```
assertion active -eosnote on -r /*
```

```
assertion active -eosnote off /tb/assert*
```

```
assertion active -eosnote on /tb/assert* /tb/cntrl/*
```

Related Topics

- [assertion action](#)
- [assertion count](#)
- [assertion enable](#)
- [assertion fail](#)
- [assertion pass](#)
- [assertion profile](#)
- [atv log](#)
- [Viewing Assertions in the Assertions Window](#)

assertion count

This command returns the sum of the assertion failure counts for the specified set of assertion directive instances. Returns a "No matches" warning if the given path does not contain any assertions.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
assertion count { -fails | -failattempts } [-lang sva | ps1 | vhdl] [-concurrent | -immediate]  
[-severity info | note | warning | error | failure | fatal] [-recursive] <path> [<path> ...]
```

Arguments

- -fails | -failattempts
(required) Controls reporting of assertion directive failures. Must be specified as the first argument to the **assertion count** command.
 - fails — Returns the sum of the given assertion directive instances that have a non-zero fail count.
 - failattempts — Returns the sum of all of the fail counts of the specified assertion directive instances.
- -lang sva | ps1 | vhdl
(optional) Specifies assertions of a specific language (SVA, PSL, or VHDL). You may specify multiple languages by using this option multiple times. If -lang is not specified, all three languages are selected.
- -concurrent | -immediate
(optional) Only concurrent assertions are selected when the -concurrent switch is used. Only immediate assertions are selected when the -immediate switch is used. These are mutually exclusive options. If neither switch is specified, both concurrent and immediate assertions are selected.
- -severity info | note | warning | error | failure | fatal
(optional) Specifies the assertion severity level. When -severity is specified only assertions with the same or higher severity are selected. If not specified, assertions of all severities are selected.
- -recursive
(optional) *For use with wildcard matching.* Specifies that the scope of the matching is to descend recursively into subregions. If omitted, the search is limited to the selected region. Applies to all paths specified in the command. You can abbreviate this switch to “-r”.
- <path> [<path> ...]
(required) A space separated list of paths, that specifies the assertions to be affected. Multiple paths and wildcards are allowed. The path specifies assertions or a design region

containing multiple assertions. Must be specified as the final argument to the **assertion count** command.

Example

```
assertion count -fails -r /
```

Related Topics

- [assertion action](#)
- [assertion active](#)
- [assertion enable](#)
- [assertion fail](#)
- [assertion pass](#)
- [assertion profile](#)
- [atv log](#)
- [Configuring Assertions](#)

assertion enable

This command enables and disables assertions and cover directives.

The **assertion enable** command introduces two levels of command function. If you use the `-force` option, this command takes precedence over any use of **assertion enable** where `-force` is not used, or over any use of the `$asserton/$assertoff/$assertkill` system tasks. With `-force`, the command is a first-level command.

Without `-force`, the command is a second-level command - that is, it is the same level as the `$asserton/$assertoff/$assertkill` system tasks. Second-level commands override each other depending on which is issued last.

The `-release` option returns the status of the assertions specified by the path to the status based on last issued second level command, whether it was issued before or after `-force`. (See Example.)

Arguments to this command are order-dependent. The `<path>` must be specified as the final argument. Please read through the argument descriptions for more information.

Syntax

```
assertion enable -on | -off | -release [-assert | -cover] [-concurrent | -immediate] [-force]  
[-lang {sva | vhdl | ps1}]... [-recursive] <path> [<path> ...]
```

Arguments

- **-on** | **-off** | **-release**

(required) Controls assertions and cover directives. Must be specified as the first argument to the **assertion enable** command.

`-on` — Enables assertions and cover directives specified by `<path>`.

`-off` — Disables

`-release` — Returns the status of the assertions (specified by `<path>`) to the status based on last issued second level command, whether it was issued before or after the use of `-force` with **assertion enable**. (See Example.)

- `-assert` | `-cover`

(optional) Specifies that the command will only apply to assertions (`-assert`) or to cover directives (`-cover`). If not specified, the command applies to both.

- `-concurrent` | `-immediate`

(optional) Specifies that the command will only apply to concurrent or to immediate assertions. If not specified, the command will apply to both.

- `-force`

(optional) Forces the command to take precedence over any other invocation of the command, or of any use of the `$asserton/$assertoff/$assertkill` system task. (See Example.)

- **-lang {sva | vhdl | ps1}**
(optional) Specifies that the command applies only to assertions and cover directives of the designated language. Can be specified multiple times to choose more than one language.
- **-recursive**
(optional) *For use with wildcard matching.* Specifies that the scope of the matching is to descend recursively into subregions. If omitted, the search is limited to the selected region. Applies to all paths specified in the command. You can abbreviate this switch to “-r”.
- **<path> [<path> ...]**
(required) A space separated list of paths, that specifies the assertions to be affected. Multiple paths and wildcards are allowed. The path specifies assertions or a design region containing multiple assertions. Must be specified as the final argument to the **assertion enable** command.

Example

- Understanding first-level and second-level commands and the use of -force and -release.

If the following command is used:

```
assertion enable -on -r assert1
```

an invocation of the \$asserton/\$assertoff/\$assertkill system task can disable the assertion named *assert1*.

If the following command is used:

```
assertion enable -on -force -r assert1
```

an invocation of the \$asserton/\$assertoff/\$assertkill system task cannot disable the *assert1* assertion.

If the following command is used:

```
assertion enable -off -r assert1
```

it has no affect on the *assert1* assertion because -force is in effect.

If the following command is used:

```
assertion enable -release -r assert1
```

the -force option is released and the *assert1* assertion is disabled due to the use of a previous -off option.

If the following command is used:

```
assertion enable -off -force -r assert1
```

the *assert1* assertion is disabled with the -force option.

If the following system task is used:

```
$asserton();
```

it has no affect right away due to the use of the -force option.

If the following command is used:

assertion enable -release -r assert1

the *assert1* assertion is enabled due to the use of \$asserton().

If the following command is used:

assertion enable -off -force assert1

the *assert1* assertion is disabled

If the following command is used:

assertion enable -release assert1

the assert1 is enabled due to previous \$asserton. The effect of -force is released.

Related Topics

- [assertion action](#)
- [assertion active](#)
- [assertion count](#)
- [assertion fail](#)
- [assertion pass](#)
- [assertion profile](#)
- [atv log](#)
- [Configuring Assertions](#)

assertion fail

This command configures simulator behavior in response to a SystemVerilog or PSL assertion failure.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
assertion fail [-action {continue | break | exit}] [-limit {none | <count>}] [-log {on | off}]  
  [-lvlog] [-recursive] <path> [<path> ...]
```

Arguments

- -action {continue | break | exit}

(optional) Specifies the action to take when an assertion fails. You can specify this option multiple times, where it applies to all paths that follow it in the command line.

continue — (default) No action taken. This is not the same as disabling an assertion since logging may still be enabled for the directive. This is the default value if you do not specify this switch.

break — Halt simulation and return to the ModelSim prompt.

exit — Halt simulation and exit ModelSim.

You can change the permanent default by setting the [AssertionFailAction](#) variable in the *modelsim.ini* file.

- -limit {none | <count>}

(optional) Sets a limit on the number of times ModelSim responds to an assertion failing.

none — No limit; failure tracking remains enabled for the duration of the simulation. This is the default behavior if you do not specify this switch.

<count> — Specify a whole number.

Once the limit is reached for a particular assertion, ModelSim disables that assertion. ModelSim continues to respond to others if their limit has not been reached. You can change the permanent default by setting the [AssertionLimit](#) variable in the *modelsim.ini* file.

- -log {on | off}

(optional) Specifies whether to write a transcript message when an assertion fails. This option may be specified multiple times; it applies to all paths that follow it in the command line.

on — Enable transcript logging. This is the default behavior if you do not specify this switch.

off — Disable transcript logging.

You can change the permanent default by setting the [AssertionFailLog](#) variable in the *modelsim.ini* file.

The **-log** argument has no effect if there is no PSL in the elaborated design, and it does not apply to SystemVerilog assertion messages coming from action blocks. Those messages are always logged and are controlled by severity system tasks (**\$fatal**, **\$error**, **\$warning** and **\$info**). You can choose to ignore these system tasks using the GUI or by setting the following variables in the *modelsim.ini* file: [IgnoreSVAError](#), [IgnoreSVAFatal](#), [IgnoreSVAInfo](#), and [IgnoreSVAWarning](#).

- **-lvlog**

(optional) Local variable values corresponding to failed assertion threads are printed to the Transcript when you run **vsim -assertdebug**. You can change the permanent default (on) by setting the [AssertionFailLocalVarLog](#) variable in the *modelsim.ini* file.

- **-recursive**

(optional) *For use with wildcard matching.* Specifies that the scope of the matching is to descend recursively into subregions. If omitted, the search is limited to the selected region. Applies to all paths specified in the command.

- **<path> [<path> ...]**

(required) A space separated list of paths, that specifies the assertions to be affected. Multiple paths and wildcards are allowed. The path specifies assertions or a design region containing multiple assertions. Must be specified as the final argument to the **assertion fail** command.

Examples

- Disable logging for assertions *a.b.c.assert__0* and *a.b.c.assert__1*. The **-log** argument applies to all paths that follow it on the command line.

```
assertion fail -log off a.b.c.assert__0 a.b.c.assert__1
```

- Disable logging for assertion *a.b.c.assert__0* but enables it for *a.b.c.assert__1*.

```
assertion fail -log off a.b.c.assert__0 -log on a.b.c.assert__1
```

- Set the failure response limit to 4 for all assertions in *mydesign*. Each assertion failure will be responded to a maximum of 4 times during the current simulation.

```
assertion fail -r / -limit 4 mydesign
```

Related Topics

- [assertion action](#)
- [assertion active](#)
- [assertion count](#)
- [assertion enable](#)
- [assertion pass](#)
- [assertion profile](#)
- [atv log](#)
- [Configuring Assertions](#)

assertion pass

This command configures simulator behavior in response to a SystemVerilog or PSL assertion pass.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
assertion pass [-log {on | off}] [-recursive] <path> [<path> ...]
```

Arguments

- -log {on | off}

(optional) Specifies whether to write a transcript message when a PSL assertion passes. This option may be specified multiple times; it applies to all paths that follow it in the command line.

on — Enable transcript logging. (default)

off — Disable transcript logging.

You can change the permanent default by setting the [AssertionPassLog](#) variable in the *modelsim.ini* file.

The **-log** argument has no effect if there is no PSL in the elaborated design, and it does not apply to SystemVerilog assertion messages coming from action blocks. Those messages are always logged and are controlled by severity system tasks (**\$fatal**, **\$error**, **\$warning** and **\$info**). You can choose to ignore these system tasks using the GUI or by setting the following variables in the *modelsim.ini* file: [IgnoreSVAError](#), [IgnoreSVAFatal](#), [IgnoreSVAInfo](#), and [IgnoreSVAWarning](#).

- -recursive

(optional) *For use with wildcard matching.* A switch that specifies that the scope of the matching is to descend recursively into subregions. If omitted, the search is limited to the selected region. Applies to all paths specified in the command.

- <path> [<path> ...]

(required) Specifies the assertions to be affected. Multiple paths are specified as a space separated list. Multiple paths and wildcards are allowed. The path specifies assertions or a design region containing multiple assertions. Must be specified as the final argument to the **assertion pass** command.

Examples

- Turn on logging for assertion *a.b.c.assert__0* but not *a.b.c.assert__1*.

```
assertion pass -log on a.b.c.assert__0 -log off a.b.c.assert__1
```

Related Topics

- [assertion action](#)
- [assertion active](#)
- [assertion count](#)
- [assertion enable](#)
- [assertion fail](#)
- [assertion profile](#)
- [atv log](#)
- [Configuring Assertions](#)

assertion profile

This command generates a fine grained profile of memory usage for assertions and cover directives including: current memory used, peak memory used (as well as the simulation run time at which it peaked), and the cumulative thread count for the assertion or cover directive.

This fine-grained memory usage profile data is reported in four columns of the Assertions Browser and Cover Directives Browser: current memory, peak memory, peak memory time, and cumulative threads.

If this command is used in the middle of the simulation, memory usage profiling starts|stops from that simulation time onwards. This command does not require that designs be compiled with +acc, or that **vsim** should be running in the **-assertdebug** mode. It can be given in fully optimized simulation run.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
assertion profile [-threadthreshold <number_of_threads>] {on | off}
```

Arguments

- -threadthreshold <number_of_threads>

(optional) Causes the assertion engine to generate messages in the transcript anytime the number of threads in an assertion or cover directive exceeds the thread threshold at a clock edge. Only functions when assertion profiling is “on.”

<number_of_threads> — Any positive integer.

- {on | off}

(required) Enables or disables the collection of assertion and cover directive memory profile data. **assertion profile on** may be given at any time during simulation. Must be specified as the final argument to the **assertion profile** command.

on — Collection of memory profile data enabled.

off — Collection of memory profile data disabled.

Example

- Thread threshold is reached.

```
assertion profile -threadthreshold 100 on
```

The transcript message produced when the threshold is reached is as follows:

```
# ** Note: Assertion thread threshold reached. Thread count = 110,  
Memory = 4.8KB  
#   Time: 215 ns   Scope: test.assert01 File: ./src/profile01.sv  
Line: 9  
# ** Note: Assertion thread threshold reached. Thread count = 110,  
Memory = 4.8KB  
#   Time: 215 ns   Scope: test.cover01 File: ./src/profile01.sv Line:  
10  
# ** Note: Assertion thread threshold reached. Thread count = 110,  
Memory = 4.8KB  
#   Time: 215 ns   Scope: test.seq File: ./src/profile01.sv Line: 3
```

Note

The message is generated at every clock edge for every assertion, cover directive, and endpoint whose current thread count is more than the threshold. It also shows the memory usage, which should be the same as that shown in the current memory column of the Assertions Browser and Cover Directives Browser.

Related Topics

- [assertion action](#)
- [assertion active](#)
- [assertion count](#)
- [assertion enable](#)
- [assertion fail](#)
- [assertion pass](#)
- [atv log](#)
- [Configuring Assertions](#)
- [GUI Elements of the Assertions Window](#)
- [GUI Elements of the Cover Directives Window](#)

atv log

This command enables or disables assertion thread viewing (ATV) for the specified assertion. Multiple assertions may be specified by their pathnames.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Prerequisites

- Use the **+acc=a** argument with the `vopt` command or **-voptargs=+acc=a** argument with the `vsim` command
- Use the **-assertdebug** argument with the `vsim` command

Syntax

```
atv log { -enable | -disable } [-asserts] [-covers] [-recursive] <path> [<path> ...]
```

Arguments

- **-enable** | **-disable**
(required) Controls assertion thread viewing for the specified assertions. Must be specified as the first argument to the **atv log** command.
 - enable — Turns on assertion thread viewing.
 - disable — Turns off assertion thread viewing.
- **-asserts**
(optional) Enables thread viewing for assertions only. Thread viewing for cover directives is not enabled. If you do not specify **-asserts** or **-covers**, the default behavior is to enable both assertions and cover directives.
- **-covers**
(optional) Enables thread viewing for cover directives only. Thread viewing for assertions is not enabled. If you do not specify **-asserts** or **-covers**, the default behavior is to enable both assertions and cover directives.
- **-recursive**
(optional) *For use with wildcard matching.* A switch that specifies that the scope of the matching is to descend recursively into subregions. If omitted, the search is limited to the selected region.
- <path> [<path> ...]
(required) A space separated list of paths, that specifies the assertions to be affected. Multiple paths and wildcards are allowed. The path specifies assertions or a design region containing multiple assertions. Must be specified as the final argument to the **atv log** command.

Related Topics

- [assertion action](#)
- [assertion active](#)
- [assertion count](#)
- [assertion enable](#)
- [assertion fail](#)
- [assertion pass](#)
- [assertion profile](#)
- [Configuring Assertions](#)
- [Viewing Assertions in the Assertions Window](#)

batch_mode

This command returns “1” if ModelSim is operating in batch mode, otherwise it returns “0.” It is typically used as a condition in an if statement.

Syntax

batch_mode

Arguments

None

Examples

Some GUI commands do not exist in batch mode. If you want to write a script that will work in or out of batch mode, you can use the **batch_mode** command to determine which command to use. For example:

```
if [batch_mode] {  
    log /*  
} else {  
    add wave /*  
}
```

Related Topics

- [Modes of Operation](#)

bd

This command deletes a breakpoint. You can delete multiple breakpoints by specifying separate information groupings on the same command line.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
bd {<filename> <line_number>}
```

```
bd {<id_number> | <label>} ...
```

Arguments

- <filename>
(required when not specifying <id_number> or <label>.) A string that specifies the name of the source file in which the breakpoint is to be deleted. The filename must match the one used previously to set the breakpoint, including whether you used a full pathname or a relative name. Must be specified as the first argument to the **bd** command.
- <line_number>
(required) A string that specifies the line number of the breakpoint to be deleted.
- <id_number> | <label>
(required when not specifying <filename>.) Specifies the identification of breakpoints using markers assigned by the **bp** command. Must be specified as the first argument to the **bd** command.
 - <id_number> — A string that specifies the identification number of the breakpoint to be deleted. The identification number is set with the **-id** argument to the **bp** command. If you are deleting a C breakpoint, the identification number will have a "c" prefix.
 - <label> — A string that specifies the label of the breakpoint to be deleted. The label is set with the **-label** switch to the **bp** command.

Examples

- Delete the breakpoint at line 127 in the source file named *alu.vhd*.

```
bd alu.vhd 127
```
- Delete the breakpoint with id# 5.

```
bd 5
```
- Delete the breakpoint with the label *top_bp*

```
bd top_bp
```
- Delete the breakpoint with id# 6 and the breakpoint at line 234 in the source file named *alu.vhd*.

bd

```
bd 6 alu.vhd 234
```

- Delete the C breakpoint with id# c.4.

```
bd c.4
```

Related Topics

- [bp](#)
- [onbreak](#)
- [C Debug](#)

bookmark add wave

This command creates a named reference to a specific zoom range and scroll position in the specified Wave window. Bookmarks are saved in the wave format file and are restored when the format file is read.

You can also interactively add a bookmark through the GUI by selecting the **Wave > Bookmarks > Bookmarks** menu item.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
bookmark add wave <label> [[<range_start> [<unit>]] [<range_end> [<unit>]] [<topindex>]]
[-window <window_name>]
```

Arguments

- <label>

(required) A string that specifies the name for the bookmark. Must be specified as the first argument to the **bookmark add wave** command.
- <range_start> [<unit>]

(optional) Specifies the beginning point of the zoom range where the default starting point is zero (0).

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <limit> and <unit> within curly braces ({}).

The complete grouping of <range_start> and <range_end> must also be enclosed in braces ({ }) or quotes (" "), for example:

```
{{100 ns} {10000 ns}}
{10000}
```
- <range_end> [<unit>]

(optional) Specifies the end point of the zoom range.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <limit> and <unit> within curly braces ({}).
- <topindex>

(optional) An integer that specifies the vertical scroll position of the window. You must specify a zoom range to specify **topindex**. The number identifies which object the window should be scrolled to. For example, specifying 20 means the Wave window will be scrolled down to show the 20th object.

bookmark add wave

- `-window <window_name>`

(optional) Specifies the window to which the bookmark will be added. If this argument is omitted, the bookmark is added in the current default Wave window.

Examples

- Add a bookmark named "foo" to the current default Wave window. The bookmark marks a zoom range from 10ns to 1000ns and a scroll position of the 20th object in the window.

```
bookmark add wave foo {{10 ns} {1000 ns}} 20
```

Related Topics

- [bookmark delete wave](#)
- [bookmark goto wave](#)
- [bookmark list wave](#)
- [write format](#)

bookmark delete wave

This command deletes bookmarks from the specified Wave window.

You can also interactively delete a bookmark through the GUI by selecting the **Wave > Bookmarks > Bookmarks** menu item.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
bookmark delete wave {<label> | -all} [-window <window_name>]
```

Arguments

- <label> | -all
(required) Controls which bookmarks to delete. Must be specified as the first argument to the **bookmark delete wave** command.
 - <label> — Specifies the name of the bookmark to delete.
 - all — Specifies that all bookmarks in the window be deleted.
- -window <window_name>
(optional) Specifies the window from which bookmark will be deleted. If this argument is omitted, bookmark(s) in the current default Wave window are deleted.
 - <window_name> — The name of the wave window containing the bookmark to be deleted.

Examples

- Delete the bookmark named "foo" from the current default Wave window.

```
bookmark delete wave foo
```
- Delete all bookmarks from the Wave window named "wave1".

```
bookmark delete wave -all -window wave1
```

Related Topics

- [bookmark add wave](#)
- [bookmark goto wave](#)
- [bookmark list wave](#)
- [write format](#)

bookmark goto wave

This command zooms and scrolls a Wave window using the specified bookmark.

You can also interactively navigate between bookmarks through the GUI by selecting the **Wave > Bookmarks > Bookmarks** menu item.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
bookmark goto wave <label> [-window <window_name>]
```

Arguments

- <label>
(required) Specifies the bookmark to go to. Must be specified as the first argument to the **bookmark goto wave** command.
- -window <window_name>
(optional) Specifies the Wave window to which the bookmark applies. Bookmarks can be used only in the windows in which they were originally created.

<window_name> — The name of the wave window containing the specified bookmark.

Related Topics

- [bookmark add wave](#)
- [bookmark delete wave](#)
- [bookmark list wave](#)
- [write format](#)

bookmark list wave

This command displays a list of available bookmarks in the Transcript window.

Syntax

```
bookmark list wave [-window <window_name>]
```

Arguments

- `-window <window_name>`
(optional) Specifies the Wave window to which the bookmark applies. Bookmarks can be used only in the windows in which they were originally created.

Related Topics

- [bookmark add wave](#)
- [bookmark delete wave](#)
- [bookmark goto wave](#)
- [write format](#)

bp

This command sets either a file-line breakpoint or returns a list of currently set breakpoints. It allows enum names, as well as literal values, to be used in condition expressions.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

A set breakpoint affects every SystemC instance in the design unless you use the **-inst <region>** argument.

Since C Debug is invoked when you set a breakpoint within a SystemC module, your C Debug settings must be in place prior to setting a breakpoint. See “[Setting Up C Debug](#)” for more information.

Note



You cannot set breakpoints when running in full optimization mode. Increase the visibility of the design by setting the **+acc** argument to **vopt**. Refer to the chapter “[Optimizing Designs with vopt](#)” in the User’s Manual for more information.

Syntax

Setting an HDL breakpoint

```
bp {<filename> <line_number>}[-appendinst] [<command>...]
    [-cond "<condition_expression>"] [-disable] [-id <id_number> | -label "<label>"]
    [-inst <region> [-inst <region> ...]]
```

Setting a C breakpoint

```
bp {-c {<function_name> | [<file_name>:]<line_number> | *0x<hex_address>}}
    [-appendinst] [<command>...] [-cond "<condition_expression>"] [-disable]
    [-id <id_number> | -label "<label>"] [-inst <region> [-inst <region> ...]]
```

Querying a breakpoint

```
bp [-query <filename> [<line_number>]]
```

Reporting all breakpoints

If you specify this command with no arguments, it returns a list of all breakpoints in the design containing information about each breakpoint. For example:

```
bp
```

returns:

```
# bp top.vhd 70;# 2
```

- o bp — an echo of the command
- o <file_name>
- o <line_number>

- # <id_number>

Arguments

- <filename>
(required to set an HDL breakpoint) Specifies the name of the source file in which to set the breakpoint. Must be specified as the first argument to the **bp** command.
- <line_number>
(required to set an HDL breakpoint) Specifies the line number where the breakpoint is to be set. Must be specified as the second argument to the **bp** command.
- **-c**
(required to set a C breakpoint) Applies the bp command and its arguments to SystemC instances in the design. Must be specified as the first argument to the **bp** command.
- <function_name> | [<file_name>:]<line_number> | *0x<hex_address>
(required) Controls the location of a C breakpoint. Must be specified as the second argument to the **bp** command.
 - <function_name> — Sets the C breakpoint at the entry to the specified function.
 - [<file_name>:]<line_number> — Sets the C breakpoint at the specified line number of the specified file. If you do not specify a file name, the breakpoint is set at the line number of the current C or SystemC file.
 - *0x<hex_address> — Sets the C breakpoint at the specified hex address.
- **-appendinst**
(optional) When specifying multiple breakpoints with **-inst**, append each instance-path condition to the earlier condition. This overrides the default behavior, in which each condition overwrites the previous one.
- **-disable**
(optional) Sets the breakpoint to a disabled state. You can enable the breakpoint later using the **enablebp** command. This command enables breakpoints by default.
- <command>...
(optional) Specifies one or more commands that are to be executed at the breakpoint. You must separate multiple commands with semicolons (;) or place them on multiple lines. Braces are required only if the string contains spaces.

Note



You can also specify this command string by choosing **Tools > Breakpoints...** from the main menu and using the **Modify Breakpoints** dialog box.

Any commands that follow a **run** or **step** command are ignored. A run or step command terminates the breakpoint sequence. This rule also applies if you use a macro within the command string.

You cannot use a [restore](#) command.

If many commands are needed after the breakpoint, you could place them in a macro file.

- `-cond "<condition_expression>"`

(optional) Specifies condition(s) that determine whether the breakpoint is hit.

"<condition_expression>" — A conditional expression that results in a true/false value. You must enclose the condition expression within quotation marks (" ").

If the condition is true, the simulation stops at the breakpoint. If false, the simulation bypasses the breakpoint. A condition cannot refer to a VHDL variable (only a signal).

The **-cond** switch re-parses expressions each time the breakpoint is hit. This allows expressions with local references to work. Condition expressions referencing items outside the context of the breakpoint must use absolute names. This is different from the behavior in previous ModelSim versions where a relative signal name was resolved at the time the **bp** command was issued, allowing the breakpoint to work even though the relative signal name was inappropriate when the breakpoint was hit.

Note



You can also specify this expression by choosing **Tools > Breakpoints...** from the main menu and using the **Modify Breakpoints** dialog box.

The condition expression can use the following operators:

Operation	Operator Syntax
equals	==, =
not equal	!=, /=
AND	&&, AND
OR	, OR

The operands may be object names, `signame'event`, or constants. Subexpressions in parentheses are permitted. The command will be executed when the expression is evaluated as TRUE or 1. The formal BNF syntax for an expression is:

```

condition ::= Name | { expression }

expression ::= expression AND relation
              | expression OR relation
              | relation

relation ::= Name = Literal
           | Name /= Literal
           | Name ' EVENT
           | ( expression )

Literal ::= '<char>' | "<bitstring>" | <bitstring>

```

The "=" operator can occur only between a Name and a Literal. This means that you cannot compare the value of two signals (for example, `Name = Name` is not valid).

You can construct a breakpoint such that the simulation breaks when a SystemVerilog Class is associated with a specific handle, or address:

```
bp <filename> <line_number> -cond "this==<class_handle>"
bp <filename> <line_number> -cond "this!=<class_handle>"
```

where you can obtain the class handle with the [examine -handle](#) command. The string "this" is a literal that refers to the specific *line_number*.

You can construct a breakpoint such that the simulation breaks when a line number is of a specific class type or extends the specified class type:

```
bp <filename> <line_number> -cond "this ISA <class_type_name>"
```

where *class_type_name* is the actual class name, not a variable.

- -id <id_number> | -label "<label>"

(optional) Attempts to assign an id number or label to the breakpoint. The command returns an error if the id number you specify is already assigned.

-id <id_number> — Any positive integer that is not already assigned.

-label "<label>" — Associates a name or label with the specified breakpoint. Adds a level of identification to the breakpoint. The label may contain special characters. Quotation marks (" ") or braces ({ }) are required only if <label> contains spaces or special characters.

Note



Id numbers for breakpoints are assigned from the same pool as those used for the [when](#) command. So even if you have not specified a given id number for a breakpoint, that number may still be used for a **when** command.

- -inst <region> [-inst <region> ...]

(optional) Sets an HDL or SystemC breakpoint so it applies only to the specified instance. To apply multiple instance-path conditions on a single breakpoint, specify **-inst <region>** multiple times. By default, this overrides the previous breakpoint condition (you can use the **-appendinst** argument to append conditions instead).

<region> — The full path to the instance specified.

Note



You can also specify this instance by choosing **Tools > Breakpoints...** from the main menu and using the **Modify Breakpoints** dialog box.

- -query <filename> [<line_number>]

(optional) Returns information about the breakpoint(s) set in the specified file. The information returned varies depending on the condition of the breakpoint(s) in the specified file. Returns a complete list of all breakpoints and whether they are enabled or not when specified without <line_number>. Returns nothing if <line_number> is not executable.

<filename> — The name of the file containing the breakpoint.

<line_number> — The line number where a breakpoint has been set.

The output contains six fields of information. For example:

```
bp -query top.vhd 70
```

returns

```
# 1 1 top.vhd 70 2 1
```

- o {1 | 0} — Indicates whether a breakpoint exists at the location.

0 — Breakpoint does not exist.

1 — Breakpoint exists.

- o 1 — always reports a 1.

- o <file_name>

- o <line_number>

- o <id_number>

- o {1 | 0} — Indicates whether the breakpoint is enabled.

0 — Breakpoint is not enabled.

1 — Breakpoint is enabled.

Examples

- List all existing breakpoints in the design, including the source file names, line numbers, breakpoint id#s, labels, and any commands that have been assigned to the breakpoints.

```
bp
```

- Set a breakpoint in the source file *alu.vhd* at line 147.

```
bp alu.vhd 147
```

- Execute the *macro.do* macro file when the breakpoint is hit.

```
bp alu.vhd 147 {do macro.do}
```

- Set a breakpoint on line 22 of *test.vhd*. When the breakpoint is hit, the values of variables *var1* and *var2* are examined. This breakpoint is initially disabled; it can be enabled with the [enablebp](#) command.

```
bp -disable test.vhd 22 {echo [exa var1]; echo [exa var2]}
```

- Set a breakpoint in every instantiation of the file *test.vhd* at line 14. When that breakpoint is executed, the Tcl command is run. This Tcl command causes the simulator to continue if the current simulation time is not 100.

```
bp test.vhd 14 {if {$now != 100} then {cont}}
```

- Set a breakpoint so that the simulation pauses whenever `clk=1` and `prdy=0`:
bp test.vhd 14 -cond "clk=1 AND prdy=0"
- Set a breakpoint with the label `top_bp`
bp top.vhd 14 -label top_bp
- Set a breakpoint for line 15 of `a.vhd`, but only for the instance `a2`:
bp a.vhd 15 -inst "/top/a2"
- Set multiple breakpoints in the source file `test.vhd` at line 14. The second instance will overwrite the conditions of the first.
bp test.vhd 14 -inst /test/inst1 -inst /test/inst2
- Set multiple breakpoints at line 14. The second instance will append its conditions to the first.
bp test.vhd 14 -inst /test/inst1 -inst /test/inst2 -appendinst
- Set a breakpoint for a specific variable of a particular class type:
set x [examine -handle my_class_var]
bp top.sv 15 -cond "this == \$x"
- List the line number and enabled/disabled status (1 = enabled, 0 = disabled) of all breakpoints in `testadd.vhd`.
bp -query testadd.vhd
- List details about the breakpoint on line 48.
bp -query testadd.vhd 48
- List all executable lines in `testadd.vhd` between lines 2 and 59.
bp -query testadd.vhd 2 59
- Sets a C breakpoint at the entry to C function **`and_gate_init`**.
bp -c and_gate_init
- Sets a C breakpoint at line 46 in the file `and_gate.c`.
bp -c and_gate.c:46
- Sets a C breakpoint at line 44 in the current C or SystemC file.
bp -c 44
- Sets a C breakpoint at hexadecimal address **`0xff130504`**.
bp -c *0xff130504
- Sets a C breakpoint for instances **`sctop.a.b`** and **`sctop.a.d`**.
bp -c -inst "sctop.a.b sctop.a.d"

bp

- Sets a C breakpoint for all instances whose name begins with **sctop.a.c**.

bp -c -inst "sctop.a.c*"

Note



Any breakpoints set in VHDL code and called by either resolution functions or functions that appear in a port map are ignored.

Related Topics

- [add button](#)
- [bd](#)
- [C Debug](#)
- [disablebp](#)
- [enablebp](#)
- [Modifying File-Line Breakpoints](#)
- [onbreak](#)
- [Optimizing Designs with vopt](#)
- [SystemC Simulation](#)
- [when](#)

cd

This command changes the ModelSim local directory to the specified directory.

This command cannot be executed while a simulation is in progress. Also, executing a **cd** command will close the current project.

Syntax

```
cd [<dir>]
```

Arguments

- <dir>
(optional) Specifies a full or relative directory path for QuestaSim to use as the local directory. If you do not specify a directory, the command changes to your home directory.

cdbg

This command provides command-line equivalents of the menu options that are available for C Debug.

For some of the commands there is a required argument "on | off". The value must be either "on" or "off." For example:

```
cdbg enable_auto_step on
cdbg stop_on_quit off
```

Syntax

```
cdbg allow_lib_step {on | off}
cdbg auto_find_bp
cdbg debug_on
cdbg enable_auto_step {on | off}
cdbg init_mode_complete
cdbg init_mode_setup
cdbg interrupt
cdbg keep_user_init_bps {on | off}
cdbg quit
cdbg refresh_source_window
cdbg set_debugger <path>
cdbg show_source_balloon {on | off}
cdbg stop_on_quit {on | off}
cdbg trace_entry_point {on | off} [<function_name>]}
```

Arguments

- allow_lib_step {on | off}
Enables stepping out from OSCI library functions. When you try to step inside OSCI library functions, C Debug automatically steps out to the last user function that was called. Note that setting this argument to "on" disables the stepping out action.
 - on — Disables stepping out from OSCI library functions.
 - off — Enables stepping out from OSCI library functions.
- auto_find_bp
Sets breakpoints on all currently known function entry points.
Equivalent to selecting **Tools > C Debug > Auto find bp**.

- `debug_on`
Enables the C Debugger.
Equivalent to selecting **Tools > C Debug > Start C Debug**.
- `enable_auto_step {on | off}`
Specifies auto-step mode enable/disable.
 - on — Enables auto-step mode.
 - off — Disables auto-step mode.Equivalent to selecting **Tools > C Debug > Enable auto step**.
- `init_mode_complete`
Instructs C Debug to continue loading the design without stopping at function calls.
Equivalent to selecting **Tools > C Debug > Complete load**. Not supported on Windows platforms.
- `init_mode_setup`
Enables initialization mode.
Equivalent to selecting **Tools > C Debug > Init mode**. Not supported on Windows platforms.
- `interrupt`
Reactivates the C debugger when stopped in HDL code.
Equivalent to selecting **Tools > C Debug > C Interrupt** or clicking the 'C Interrupt' toolbar button.
- `keep_user_init_bps {on | off}`
Specifies whether breakpoints set during initialization mode are retained after the design finishes loading.
 - on — Enables retention of breakpoints after design loading is finished.
 - off — Disables retention of breakpoints after design loading is finished.Equivalent to toggling the 'Keep user init bps' button in the C Debug setup dialog.
- `quit`
Closes the C Debugger.
Equivalent to selecting **Tools > C Debug > Quit C Debug**.
- `refresh_source_window`
Re-opens a C source file if you close the Source window inadvertently while stopped in the C debugger.
Equivalent to selecting **Tools > C Debug > Refresh**.

- `set_debugger <path>`
Sets the path to your **gdb** installation.
`<path>` — The complete pathname to the **gdb** executable. For example:
`cdbg set_debugger_path /usr/bin/gdb`
Equivalent to selecting **Tools > C Debug > C Debug Setup** and entering a custom path.
- `show_source_balloon {on | off}`
Toggles the source balloon popup.
`on` — Enables balloon popup.
`off` — Disables balloon popup.
Equivalent to toggling the 'Show balloon' button on the C Debug setup dialog.
- `stop_on_quit {on | off}`
Toggles debugging capability when the simulator is exiting.
`on` — Enables debugging when the simulator is exiting.
`off` — Disables debugging when the simulator is exiting.
Equivalent to toggling the 'Stop on quit' button on the C Debug setup dialog.
- `trace_entry_point {on | off} [<function_name>]`
Helps debug an FLI/PLI application when a design is loaded with **vsim** **-trace_foreign**. ModelSim stops at a C breakpoint each time a named FLI or PLI function is called from your application. Once at the breakpoint, use the **tb** and **pop** commands to investigate the C code at the place the function was called.
`on` — Enables debugging.
`off` — Disables debugging.
`<function_name>` — An FLI or PLI function call.

Related Topics

- [Debugging Functions During Elaboration](#)
- [Debugging Functions when Quitting Simulation](#)
- [Finding Function Entry Points with Auto Find bp](#)
- [Identifying All Registered Function Calls](#)

change

This command modifies the value of a:

- VHDL constant, generic, or variable
- Verilog register or variable
- C variable if running C Debug

You cannot use this command on generics or parameters if you optimized the design, unless you used the `+floatgenerics` or `+floatparameters` switches. These switches allow the generics and parameters to remain floating after the optimization. Refer to the section "[Optimizing Parameters and Generics](#)" in the User's Manual for more information.

Syntax

```
change <variable> <value>
```

Arguments

- <variable>

(required) A string that specifies the name of an object. The name can be a full hierarchical name or a relative name, where a relative name is relative to the current environment.

Wildcards are not permitted.

The following sections list supported objects:

- VHDL
 - Scalar variable, constant, or generics of all types except FILE.
Generates a warning when changing a VHDL constant or generic. You can suppress this warning by setting the TCL variable `WarnConstantChange` to 0 or in the `[vsim]` section of the `modelsim.ini` file.
 - Scalar subelement of composite variable, constant, and generic of all types except FILE.
 - One-dimensional array of enumerated character types, including slices.
 - Access type. An access type pointer can be set to "null"; the value that an access type points to can be changed as specified above.
- Verilog
 - Parameter.
 - Register or memory.
 - Integer, real, realtime, time, and local variables in tasks and functions.
 - Subelements of register, integer, real, realtime, and time multi-dimensional arrays (all dimensions must be specified).

- Bit-selects and part-selects of the above except for objects whose basic type is real.
- C
 - Scalar C variables of type int, char, double, or float.
 - Individual fields of a C structure.
 - SystemC primitive channels are not supported.

The name can be a full hierarchical name or a relative name. A relative name is relative to the current environment. Wildcards cannot be used.

- <value>
(required) Defines a value for <variable>. The specified value must be appropriate for the type of the variable. You must place <value> within quotation marks (“ ”) or curly braces ({ }) if it contains spaces .

Note



The initial type of <variable> determines the type of value that it can be given. For example, if <variable> is initially equal to 3.14 then only real values can be set on it. Also note that changing the value of a parameter or generic will not modify any design elements that depended on the parameter or generic during elaboration (for example, sizes of arrays).

Examples

- Change the value of the variable *count* to the hexadecimal value FFFF.
change count 16#FFFF
- Change the value of the element of *rega* that is specified by the index (i.e., 16).
change {rega[16]} 0
- Change the value of the set of elements of *foo* that is specified by the slice (i.e., 20:22).
change {foo[20:22]} 011
- Set the value of *x* (type double) to 1.5.
change x 1.5
- Set the value of structure member *a1.c1* (type int) to 0.
change a1.c1 0
- Set *val_b* (type char *) to point to the string *my_string*.
change val_b my_string
- Set *val_b* (type char *) to point to the string *my string*. Since there is a space in the value, it must be enclosed by quotation marks or curly braces.
change val_b "my string"

-
- Set the Verilog register *file_name* to "test2.txt". Note that the quote marks are escaped with `\'`.

```
change file_name \'test2.txt\'
```

- Set the time value of the mytimegeneric variable to 500 ps. The time value is enclosed by curly braces (or quotation marks) because of the space between the value and the units.

```
change mytimegeneric {500 ps}
```

Related Topics

- [force](#)

change_menu_cmd

This command changes the command to be executed for a specified menu item label, in the specified menu, in the specified window.

The menu path and label must already exist for this command to function. Returns nothing.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
change_menu_cmd <window_name> <menu_path> <label> <cmd>
```

Arguments

The arguments <window_name>, <menu_path>, <cmd>, and <label> are order dependent and must be entered as shown in the Syntax section.

- <window_name>
(required) Specifies the Tk path of the window containing the menu. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).
- <menu_path>
(required) Specifies the name of an existing Tk menu widget plus any submenu path.
- <label>
(required) Specifies the current label on the menu item.
- <cmd>
(required) Specifies the new Tcl command to be executed when selected.

Related Topics

- [add_menu](#)
- [add_menub](#)
- [add_menuitem](#)
- [add_separator](#)
- [add_submenu](#)

check contention add

This command enables contention checking for the specified nodes.

The allowed nodes are Verilog nets and VHDL signals of types `std_logic` and `std_logic_vector`. This command ignores any other node types or nodes that do not have multiple drivers.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
check contention add {[-in] [-out] [-inout] | [-ports]} [-internal] [-r] <node_name>...
```

Arguments

- `-in`
 (optional) Enables checking on nodes of mode IN.
- `-inout`
 (optional) Enables checking on nodes of mode INOUT.
- `-out`
 (optional) Enables checking on nodes of mode OUT.
- `-ports`
 (optional) Enables checking on nodes of modes IN, OUT, or INOUT. Default behavior if no arguments are specified.
- `-internal`
 (optional) Enables checking on internal (non-port) objects. Default behavior if no arguments are specified.
- `-r`
 (optional) Specifies that contention checking is enabled recursively into subregions. If omitted, contention check enabling is limited to the current region.
- `<node_name>...`
 (required) Specifies the name of a node. Must be specified as the final argument to the **check contention add** command.

Description

Bus contention checking detects bus fights on nodes that have multiple drivers. A bus fight occurs when two or more drivers drive a node with the same strength and that strength is the strongest of all drivers currently driving the node. The following table provides some examples for two drivers driving a `std_logic` signal:

driver 1	driver 2	fight
Z	Z	no

driver 1	driver 2	fight
0	0	yes
1	Z	no
0	1	yes
L	1	no
L	H	yes

Detection of a bus fight results in an error message specifying the node and its drivers' current driving values. If a node's drivers later change value and the node is still in contention, a message is issued giving the new values of the drivers. A message is also issued when the contention ends. The bus contention checking commands can be used on VHDL and Verilog designs.

Related Topics

- [check contention config](#)
- [check contention off](#)

check contention config

This command allows you to write checking messages to a file. By default, any messages display on your screen.

You may also configure the contention time limit.

Syntax

```
check contention config [-file <filename>] [-time <limit>[<unit>]]
```

Arguments

- `-file <filename>`
(optional) Specifies to write contention messages to a specified file. When you specify this switch, check contention messages will not be displayed to the screen.
`<filename>` — The name of a file where contention messages will be saved.
- `-time <limit>[<unit>]`
(optional) Specifies a time limit that a node may be in contention. Contention is detected if a node is in contention for as long as or longer than the limit.
`<limit>` — Any non-negative integer where the default is 0.
`<unit>` — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting `<unit>`. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If `<unit>` is specified, you must enclose `<limit>` and `<unit>` within curly braces (`{ }`).

Related Topics

- [check contention add](#)
- [check contention off](#)

check contention off

This command disables contention checking for the specified nodes.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
check contention off [-all] {[-in] [-out] [-inout] | [-ports]} [-internal] [-r] <node_name> ...
```

Arguments

- -all
(optional) Disables contention checking for all nodes that have checking enabled.
- -in
(optional) Disables checking on nodes of mode IN.
- -out
(optional) Disables checking on nodes of mode OUT.
- -inout
(optional) Disables checking on nodes of mode INOUT.
- -ports
(optional) Disables checking on nodes of modes IN, OUT, or INOUT.
- -internal
(optional) Disables checking on internal (non-port) objects.
- -r
(optional) Specifies that contention checking is disabled recursively into subregions. If omitted, contention check disabling is limited to the current region.
- <node_name> ...
(required) Specifies the named node(s). Must be specified as the final argument to the **check contention off** command.

Related Topics

- [check contention add](#)
- [check contention config](#)

check float add

This command enables float checking for the specified nodes.

The allowed nodes are Verilog nets and VHDL signals of type `std_logic` and `std_logic_vector` (other types are silently ignored).

You can set a time limit (the default is zero) for float checking using the **-time <limit>** argument to the [check float config](#) command. If you choose to modify the limit, you should do so prior to invoking any **check float add** commands.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
check float add {[-in] [-out] [-inout] | [-ports]} [-internal] [-r] <node_name> ...
```

Arguments

- **-r**
(optional) Specifies that float checking is enabled recursively into subregions. If omitted, float check enabling is limited to the current region.
- **-in**
(optional) Enables checking on nodes of mode IN.
- **-out**
(optional) Enables checking on nodes of mode OUT.
- **-inout**
(optional) Enables checking on nodes of mode INOUT.
- **-internal**
(optional) Enables checking on internal (non-port) objects.
- **-ports**
(optional) Enables checking on nodes of modes IN, OUT, or INOUT.
- **<node_name> ...**
(required) Enables checking for the named node(s). Must be specified as the final argument to the **check float add** command.

Description

Bus float checking detects nodes that are in the high impedance state for a time equal to or exceeding a user-defined limit. This is an error in some technologies. Detection of a float violation results in an error message identifying the node. A message is also issued when the float violation ends. The bus float checking commands can be used on VHDL and Verilog designs.

Related Topics

- [check contention config](#)
- [check float off](#)

check float config

This command allows you to write checking messages to a file (messages display on your screen by default). You may also configure the float time limit.

Syntax

```
check float config [-file <filename>] [-time <time> [<unit>]]
```

Arguments

- `-file <filename>`
(optional) Writes float messages to a file. If this option is selected, the messages are not displayed to the screen.
`<filename>` — Specifies the name of the file that float messages will be saved to.
- `-time <time><unit>`
(optional) Specifies a length of time that a node may be floating. An error is detected if a node is floating for as long as or longer than the limit. Note that you should configure the time limit prior to invoking any [check float add](#) commands.
`<time>` — Any non-negative integer where the default is 0.
`<unit>` — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting `<unit>`. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If `<unit>` is specified, you must enclose `<time>` and `<unit>` within curly braces (`{ }`).

Related Topics

- [check float add](#)
- [check float off](#)

check float off

This command disables float checking for the specified nodes.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
check float off [-all] {[-in] [-out] [-inout] | [-ports]} [-internal] [-r] <node_name> ...
```

Arguments

- -all
(optional) Disables float checking for all nodes that have checking enabled.
- -in
(optional) Disables checking on nodes of mode IN.
- -out
(optional) Disables checking on nodes of mode OUT.
- -inout
(optional) Disables checking on nodes of mode INOUT.
- -ports
(optional) Disables checking on nodes of modes IN, OUT, or INOUT.
- -internal
(optional) Disables checking on internal (non-port) objects.
- -r
(optional) Specifies that float checking is disabled recursively into subregions. If omitted, float check disabling is limited to the current region.
- <node_name> ...
(required) Disables checking for the named node(s). Must be specified as the final argument to the **check float off** command.

Related Topics

- [check float add](#)
- [check float config](#)

check stable off

This command disables stability checking.

You may later enable it with [check stable on](#), and meanwhile, the clock cycle numbers and boundaries are still tracked.

Syntax

```
check stable off
```

Arguments

- None

Related Topics

- [check stable on](#)

check stable on

This command enables stability checking on the entire design.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
check stable on [-period <time>[<unit>]] [-file <filename>] [-strobe <time> [<unit>]]
```

Arguments

- -period <time> [<unit>]

(required the first time you invoke the **check stable on** command.) Specifies the clock period (which is assumed to begin at the time the **check stable on** command is issued). Must be specified as the first argument to the **check stable on** command.

<time> — Any non-negative integer.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).

This argument is not required if you later enable checking after it was disabled with the [check stable off](#) command.

- -file <filename>

(optional) Writes the error messages to a specified file. If this option is selected, the messages are not displayed to the screen.

<filename> — Specifies the name of the file that error messages will be saved to.

- -strobe <time> [<unit>]

(optional) Specifies the elapsed time within each clock cycle that the stability check is performed. The default strobe time is the time specified in **-period**. If the strobe time falls on a period boundary, then the check is actually performed one timestep earlier. Normally the strobe time is specified as less than or equal to **-period**, but if it is greater than **-period**, then the check will skip cycles.

<time> — Any non-negative integer.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).

Description

Design stability checking detects when circuit activity has not settled within a period you define for synchronous designs. You specify the clock period for the design and the strobe time within the period during which the circuit must be stable. A violation is detected and an error message

is issued if there are pending driver events at the strobe time. The message identifies the driver that has a pending event, the node that it drives, and the cycle number. The design stability checking commands can be used on VHDL and Verilog designs.

Examples

- Performs a stability check 99 ps into each even numbered clock cycle (cycle numbers start at 1).

check stable on -period "100 ps" -strobe "199 ps"

Related Topics

- [check stable off](#)

checkpoint

This command saves the state of your simulation, including:

- *modelsim.ini* settings.
- the simulation kernel state.
- the *vsim.wlf* file.
- the list of the design objects shown in the List and Wave windows.
- the file pointer positions for files opened under VHDL and the Verilog **\$fopen** system task.
- the states of foreign architectures.
- VCD output.
- Toggle statistics (see the [toggle report](#) command).

However, it does not save the following:

- Changes you made interactively while running [vsim](#) are not saved; for example, macros, virtual objects, command-line interface additions like user-defined commands, and states of graphical user interface
- Transactions

Once saved, a checkpoint file may be used with the [restore](#) command during the same simulation to restore the simulation to a previous state. A **vsim** session may also be started with a checkpoint file by using the **vsim -restore** command.

Compression of the checkpoint file is controlled by the [CheckpointCompressMode](#) variable in the *modelsim.ini* file.

If a checkpoint occurs while ModelSim is writing a VCD file, the entire VCD file is copied into the checkpoint file. Since VCD files can be very large, it is possible that disk space problems could occur. Consequently, ModelSim issues a warning in this situation.

Checkpoint files are platform dependent, therefore you cannot checkpoint on one platform and restore on another.

If checkpointing DPI code that works with heap memory, use `mti_Malloc()` rather than `raw malloc()` or `new`. Any memory allocated with `mti_Malloc()` is guaranteed to be restored correctly. Any memory allocated with `raw malloc()` will not be restored correctly, and simulator crashes can result.

Syntax

```
checkpoint <filename>
```

Arguments

- `<filename>`
(required) Specifies the name of the checkpoint file.

Related Topics

- [restore](#)
- [restart](#)
- [vsim](#)
- [Checkpointing and Restoring Simulations](#)

compare add

This command creates an object that is a comparison between signals in a reference design against signals in a test design. You can specify whether to compare two signals, all signals in the region, just ports or a subset of ports. Constant signals such as parameters and generics are ignored.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Refer to [Waveform Compare](#) for a general overview of waveform comparisons.

The names of the added comparison objects take the form:

`<path>/refSignalName<>testSignalName\`

If you compare two signals from different regions, the signal names include the uncommon part of the path. [Table 2-2](#) shows how comparisons work between specified reference objects and test objects.

Table 2-2. Comparing Reference Objects to Test Objects

Reference object	Test object	Result
signal	signal	compare the two signals
signal	region	compare a signal with a name matching the reference signal in the specified test region
region	region	compare all matching signals in both regions
glob expression	signal	legal only if the glob expression selects only one signal
glob expression	region	compare all signals matching the glob expression that match signals in the test region

The **compare add** command supports arguments that specify how each signal state matches `std_logic` or Verilog values (for example, **-vhdlmatches**, see below). Since state matching can also be set on a global basis with the [compare options](#) command or `PrefCompare()` Tcl variables, ModelSim follows state match settings in the following order:

1. Use local matching values specified when the comparison was created using **compare add** or subsequently configured using [compare configure](#).
2. If no local values were set, use global matching values set with the **compare options** command.
3. If no compare options were set, use default matching values specified by `PrefCompare` Tcl variables.

Syntax

```
compare add [-all] {[[-in] [-inout] [-out] | [-ports]] [-internal] [-label <label>] [-list] [-rebuild]
[-recursive] [-separator <string>] [<testPath>] [-tol <time> [<unit>]]
[-tolLead <time> [<unit>]] [-tolTrail <time> [<unit>]] [-verbose]
[-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}]
[-vlogmatches {<ref-logic-value>=<test-logic-value>:...}] [-wave] [-wavepane <n>]
[-when {"<expression>"}] [-win <wname>] [-nowin] -clock <name> <referencePath>
```

Arguments

- -all
(optional) Specifies comparison of all signals.
- -help
(optional) Lists the description and syntax for the **compare add** command in the Transcript window.
- -in
(optional) Specifies comparison of ports of mode IN. Multiple modes can be specified in the same command.
- -inout
(optional) Specifies comparison of ports of mode INOUT. Multiple modes can be specified in the same command.
- -out
(optional) Specifies comparison of ports of mode OUT. Multiple modes can be specified in the same command.
- -ports
(optional) Specifies comparison of all ports. This switch has the same effect as specifying **-in**, **-out**, and **-inout** together.
- -internal
(optional) Specifies comparison of internal (non-port) signals. Multiple modes can be specified in the same command.
- -label <label>
(optional) Specifies a name for the comparison when it is displayed in the Wave window.
 <label> — The specified name for the comparison.
- -list
(optional) Causes specified comparisons to be displayed in the default List window.
- -rebuild
(optional) Rebuilds a fragmented bus in the test design region and compares it with the corresponding bus in the reference design region. If a signal is found having the same name

as the reference signal, the **-rebuild** option is ignored. When rebuilding the test signal, the name of the reference signal is used as the wildcard prefix.

- **-recursive**

(optional) Specifies that signals should also be selected in all nested subregions, and subregions of those, and so forth.

- **-separator <string>**

(optional) Used with the **-rebuild** argument. Specifies a separator to be inserted between a base bus name and the bit indication. When a bus has been broken into bits (bit blasted) by a synthesis tool, ModelSim expects a separator between the base bus name and the bit indication. For example, the signal "mybus" might be broken down into "mybus_0", "mybus_1", etc.

<string> — Specifies the character(s) to be used as a separator between the base bus name and the bit indication where the default is an underline “_”.

For example, the signal "mybus" might be broken down into "mybus_0", "mybus_1", etc.

- **<testPath>**

(optional) Specifies an absolute or relative path to the test signal or region. Cannot be a glob expression. If omitted, the test path defaults to the same path as **<referencePath>** except for the dataset name.

- **-tol <time> [<unit>]**

(optional) Specifies the maximum amount of time a test signal edge is allowed to lead or trail a reference edge in an asynchronous comparison.

<time> — Any non-negative integer where the default is 0.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- **-tolLead <time> [<unit>]**

(optional) Specifies the maximum amount of time a test signal edge is allowed to lead a reference edge in an asynchronous comparison.

<time> — Any non-negative integer where the default is 0.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

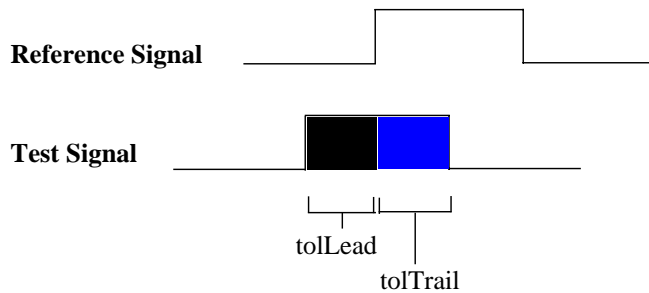
- **-tolTrail <time> [<unit>]**

(optional) Specifies the maximum amount of time a test signal edge is allowed to trail a reference edge in an asynchronous comparison.

<time> — Any non-negative integer where the default is 0.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).

Graphical representation of tolLead and tolTrail



- -verbose

(optional) Prints information in the Transcript window confirming the signals selected for comparison and any type conversions employed.

- -vhdlmatches {<ref-logic-value>=<test-logic-value>:...}

(optional) Specifies how VHDL signal states in the reference dataset should match values in the test dataset. Values are specified in a colon-separated list of match values. For example:

-vhdlmatches {X=XUD:Z=ZD:1=1HD}

Default is:

{U=UWXD:X=UWXD:0=0LD:1=1HD:Z=ZD:W=UWXD:L=0LD:H=1HD:D=UX01ZWLHD}

The 'D' character represents the '-' "don't care" std_logic value.

- -vlogmatches {<ref-logic-value>=<test-logic-value>:...}

(optional) Specifies how Verilog signal states in the reference dataset should match values in the test dataset. Values are specified in a colon-separated list of match values. For example:

-vlogmatches {0=0:1=1:Z=Z}

Default is:

{0=0:1=1:Z=Z:X=X}.

- -wave

(optional) Specifies that compare signals be added automatically to the default Wave window. (default)

- -wavepane <n>

(optional) Specifies the pane of the Wave window in which the differences will be placed.

<n> — A positive integer corresponding to the pane of the Wave window. Wave window panes are numbered in sequence starting with 1 for the initial wave window.

- -when {"<expression>"}

(optional) Specifies a conditional expression that must evaluate to "true" or "1" for differences to be reported. The expression is evaluated at the start of an observed difference. See [GUI_expression_format](#) for legal expression syntax.

"<expression>" — A conditional expression that results in a true/false value. You must enclose the condition expression within quotation marks (" ").

- -win <wname>

(optional) Adds objects to a specified window. Used to specify a particular window when multiple instances of that window type exist.

<wname> — The name of the window to which objects are to be added.

- -nowin

(optional) Specifies that compare signals shouldn't be added to any window, where the default is to add compare signals to the default Wave window. See **-wave**.

- -clock <name>

(required for clocked comparisons; not used for asynchronous comparisons.) Specifies the clock definition to use when sampling the specified regions.

<name> — Specifies the name of the clock definition.

- <referencePath>

(required) Specifies either an absolute or relative path to the reference signal or region, or a glob expression. Relative paths are relative to the current context of the reference dataset. If you specify a glob expression, it will match signals only in the containing context. Must be specified as the final argument to the **compare add** command.

Examples

- Select signals in the reference and test dataset top region according to the default mode. Uses asynchronous comparison with the default tolerances. Assumes that the top regions of the reference and test datasets have the same name and contain the same signals with the same names.

```
compare add /*
```

- Select port signals of instance *.test_ringbuf.ring_inst* in both datasets to be compared and sampled on strobe *myclock10*.

```
compare add -port -clock myclock10 gold:.test_ringbuf.ring_inst
```

- Select all signals in the *cpu* region to be compared asynchronously using the default tolerances. Requires that the reference and test relative hierarchies and signal names within the *cpu* region be identical, but they need not be the same above the *cpu* region.

```
compare add -r gold:/top/cpu test:/testbench/cpu
```


- Specify that signal *gold:.top.s1* should be sampled at *clock12* and compared with *test:.top.s1*, also sampled at *clock12*.

compare add -clock clock12 gold:.top.s1

- Specify that signal *gold:/asynch/abc/s1* should be compared asynchronously with signal *sim:/flat/sigabc* using a leading tolerance of 3 ns and a trailing tolerance of 5 ns.

compare add -toLead {3 ns} -toTrail {5 ns} gold:/asynch/abc/s1 sim:/flat/sigabc

- Cause signals *test:.counter2.cnt_dd* to be rebuilt into bus *test:.counter2.cnt[...]* and compared against *gold:.counter1.count*.

compare add -rebuild gold:.counter1.count test:.counter2.cnt

Related Topics

- [compare add](#)
- [compare annotate](#)
- [compare clock](#)
- [compare configure](#)
- [compare continue](#)
- [compare delete](#)
- [compare end](#)
- [compare info](#)
- [compare list](#)
- [compare options](#)
- [compare reload](#)
- [compare reset](#)
- [compare run](#)
- [compare savediffs](#)
- [compare saverules](#)
- [compare see](#)
- [compare start](#)
- [compare stop](#)
- [compare update](#)
- [Waveform Compare](#)

compare annotate

This command either flags a comparison difference as "ignore" or adds a text string annotation to the difference. The text string appears when the difference is viewed in info popups or in the output of a compare info command.

Some of the arguments for this command are order-dependent. Please read through the arguments for further information.

Syntax

```
compare annotate [-ignore] [-noignore] [-text <message>] <idNum1> [<idNum2>...]
```

Arguments

- **-ignore**
(optional) Flags the specified difference as "ignore."
- **-noignore**
(optional) Undoes a previous **-ignore** argument.
- **-text <message>**
(optional) Adds a text string annotation to the difference that is shown wherever the difference is viewed.
- **<idNum1> <idNum2>...**
(required) Identifies the difference number to annotate. You can obtain a difference number by using the [compare start](#) command or by placing the cursor over the logged difference in the Wave window and reading the popup dialog. Difference numbers are ordered by time of the difference start, but there may be more than one difference starting at a given time. Multiple idNumbers are specified as a space separated list. Must be specified as the final argument to **compare annotate**.

If you specify this argument with no other arguments, it returns the current annotations recorded for the specified IDs. For example:

```
compare annotate 40
```

returns:

```
# Diff 40: -ignore -text "This is a not a critical problem."
```

- Diff 40— an echo of the specified comparison difference.
- The current setting of **-ignore** or **-noignore**.
- **-text** “<text string saved to the specified difference>” If no text has been saved to the difference, an empty set of quotation marks (“”) is returned.

Examples

- Flag difference numbers 1, 2, and 10 as "ignore."

compare annotate -ignore 1 2 10

- Annotate difference number 12 with the message "THIS IS A CRITICAL PROBLEM."

compare annotate -text "THIS IS A CRITICAL PROBLEM" 12

Related Topics

- [compare add](#)
- [compare info](#)
- [Waveform Compare](#)

compare clock

This command defines a clock that can then be used for clocked-mode comparisons. In clocked-mode comparisons, signals are sampled and compared only at or just after an edge on some signal.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
compare clock [-delete] [-offset <delay><unit>] [-rising | -falling | -both]
              [-when {<expression>}] {<clock_name> <signal_path>}
```

Arguments

- **-delete**
(optional) Deletes an existing compare clock.
- **-offset <delay>[<unit>]**
(optional) Specifies a time value for delaying the sample time beyond the specified signal edge.
 - <delay> — Any non-negative integer where the default is 0.
 - <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <delay> and <unit> within curly braces ({}).
- **-rising**
(optional) Specifies that the rising edge of the specified signal should be used. (default)
- **-falling**
(optional) Specifies that the falling edge of the specified signal should be used. The default is rising.
- **-both**
(optional) Specifies that both the rising and the falling edge of the specified signal should be used. The default is rising.
- **-when {"<expression>"}**
(optional) Specifies a conditional expression that must evaluate to "true" or "1" for that clock edge to be used as a strobe. The expression is evaluated at the time of the clock edge, rather than after the delay has been applied. See [GUI_expression_format](#) for legal expression syntax.
 - "<expression>" — A conditional expression that results in a true/false value. You must enclose the condition expression within quotation marks ("").

- `<clock_name>`
(required) A name for this clock definition. This name will be used with the [compare add](#) command when doing a clocked-mode comparison. Must precede the `<signal_path>` argument.
- `<signal_path>`
(required) A full path to the signal whose edges are to be used as the strobe trigger. Must be specified as the final argument to the **compare clock** command.

Examples

- Define a clocked compare strobe named "strobe" that samples signals on the rising edge of signal `gold:.top.clock`.

```
compare clock -rising strobe gold:.top.clock
```

Define a clocked compare strobe named "clock12" that samples signals 12 ns after the rising edge of signal `gold:/mydesign/clka`.

```
compare clock -rising -delay {12 ns} clock12 gold:/mydesign/clka
```

Related Topics

- [compare add](#)
- [Waveform Compare](#)

compare configure

This command modifies options for compare signals and regions. The modified options are applied to all objects in the specified compare path.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
compare configure [-clock <name>] [-recursive] [-tol <time>[<unit>]]  
  [-tolLead <time>[unit]] [-tolTrail <time>[unit]]  
  [-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}]  
  [-vlogmatches {<ref-logic-value>=<test-logic-value>:...}] [-when {"<expression>"}]  
  <comparePath>
```

Arguments

- **-clock <name>**
(optional) Changes the strobe signal for the comparison. If the comparison is currently asynchronous, it will be changed to clocked. This switch may not be used with the **-tol**, **-tolLead**, and **-tolTrail** options.

 <name>
- **-recursive**
(optional) Specifies that signals should also be selected in all nested subregions, and subregions of those, etc.
- **-tol <time>[<unit>]**
(optional) Specifies the default maximum amount of time the test signal edge is allowed to trail or lead the reference edge in an asynchronous comparison.

 <time> — Any non-negative integer where the default is 0.

 <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).
- **-tolLead <time>[unit]**
(optional) Specifies the maximum amount of time a test signal edge is allowed to lead a reference edge in an asynchronous comparison.

 <time> — Any non-negative integer where the default is 0.

 <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).

- `-tolTrail <time>[unit]`
 (optional) Specifies the maximum amount of time a test signal edge is allowed to trail a reference edge in an asynchronous comparison.
 <time> — Any non-negative integer where the default is 0.
 <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).
- `-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}`
 (optional) Specifies how VHDL signal states in the reference dataset should match values in the test dataset. Values are specified in a colon-separated list of match values. For example:
`-vhdlmatches {X=XUD:Z=ZD:1=1HD}`
 Default is:
`{U=UWXD:X=UWXD:0=0LD:1=1HD:Z=ZD:W=UWXD:L=0LD:H=1HD:--=UX01ZWLHD}`
- `-vlogmatches {<ref-logic-value>=<test-logic-value>:...}`
 (optional) Specifies how Verilog signal states in the reference dataset should match values in the test dataset. Values are specified in a colon-separated list of match values. For example:
`-vlogmatches {0=0:1=1:Z=Z}`
 Default is:
`{0=0:1=1:Z=Z:X=X}`
- `-when {"<expression>"}`
 (optional) Specifies a conditional expression that must evaluate to "true" or "1" for differences to be reported. The expression is evaluated at the start of an observed difference. See [GUI_expression_format](#) for legal expression syntax.
 "<expression>" — A conditional expression that results in a true/false value. You must enclose the condition expression within quotation marks ("").
- `<comparePath>`
 (required) Identifies the path of a compare signal, region, or glob expression. Must be specified as the final argument to the **compare configure** command.

Related Topics

- [compare add](#)
- [Waveform Compare](#)

compare continue

This command is used to continue with comparison difference computations that were suspended using the **compare stop** button or **Control-C**. If the comparison was not suspended, **compare continue** has no effect.

Syntax

compare continue

Arguments

- None

Related Topics

- [compare stop](#)
- [Waveform Compare](#)

compare delete

This command deletes a comparison object from the currently open comparison.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
compare delete [-recursive] {<objectPath>}
```

Arguments

- `-recursive`
(optional) Deletes a region recursively.
- {<objectPath>}
(required) Path to the comparison object to be deleted (e.g., {*compare:/top^clk<>clk*}). The comparison object must be "escaped" correctly so the braces '}' and trailing space are required. Must be specified as the final argument to the **compare delete** command.

Related Topics

- [compare add](#)
- [Waveform Compare](#)

compare end

This command closes the active comparison without saving any information.

Syntax

```
compare end
```

Arguments

- None

Related Topics

- [compare add](#)
- [Waveform Compare](#)

compare info

This command lists the results of the comparison in the Main window transcript. To save the information to a file, use the **-write** argument.

Syntax

```
compare info [-all] [-count] [-primaryonly] [-signals] [-secondaryonly]
  [<startNum> [<endNum>]] [-summary] [-write <filename>]
```

Arguments

- **-all**
(optional) Lists all differences (even those marked as "ignore") in the output. By default, ignored differences are not listed in the output of a compare info command.
- **-count**
(optional) Returns the total number of primary differences found.
- **-primaryonly**
(optional) Lists only differences on individual bits, ignoring aggregate values such as a bus.
- **-signals**
(optional) Returns a Tcl list of compare signal names that have at least one difference.
- **-secondaryonly**
(optional) Lists only aggregate value differences such as a bus, ignoring the individual bits.
- **<startNum> [<endNum>]**
(optional) Specifies the difference numbers to start and end the list with. If omitted, ModelSim starts the listing with the first difference and ends it with the last. If just **endNum** is omitted, ModelSim ends the listing with the last difference.
- **-summary**
(optional) Lists only summary information.
- **-write <filename>**
(optional) Saves the summary information to <filename> rather than the Main window transcript. Optional.

Examples

- List all errors in the Main window transcript.
compare info
- List only an error summary in the Main window transcript.
compare info -summary
- Write errors 20 through 50 to the file *myerrorfile*.

compare info -write myerrorfile 20 50

Related Topics

- [compare add](#)
- [compare annotate](#)
- [Waveform Compare](#)

compare list

Displays in the Transcript window a list of all the **compare add** commands currently in effect.

Syntax

```
compare list [-expand]
```

Arguments

- -expand
(optional) Expands groups specified by the compare add command to individual signals.

Related Topics

- [compare add](#)
- [Waveform Compare](#)

compare options

This command sets defaults for various waveform comparison commands. Those defaults are used when other compare commands are invoked during the current session. To set defaults permanently, edit the appropriate PrefCompare() Tcl variable.

Refer to “[Simulator GUI Preferences](#)” for details.

If no arguments are used, compare options returns the current setting for all options. If one option is given that requires a value, and if that value is not given, compare options returns the current value of that option.

Syntax

```
compare options [-addwave] [-all] {[[-in] [-inout] [-out] | [-ports]]} [-internal] [-noaddwave]
[-ignoreVlogStrengths] [-noignoreVlogStrengths] [-maxsignal <n>] [-maxtotal <n>]
[-listwin <name>] [-separator <string>] [-tol <time>[<unit>]] [-tolLead <time>[<unit>]]
[-tolTrail <time>[<unit>]] [-track] [-notrack]
[-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}]
[-vlogmatches {<ref-logic-value>=<test-logic-value>:...}] [-wavepane <n>]
[-wavewin <name>]
```

Arguments

- **-addwave**
(optional) Specifies that new comparison objects are added automatically to the Wave window. (default) You can specify that objects aren't added automatically using the **-noaddwave** argument. Related Tcl variable is PrefCompare(defaultAddToWave).
- **-all**
(optional) Specifies comparison of all signals.
- **-in**
(optional) Specifies comparison of ports of mode IN. Multiple modes can be specified in the same command.
- **-inout**
(optional) Specifies comparison of ports of mode INOUT. Multiple modes can be specified in the same command.
- **-ignoreVlogStrengths**
(optional) Specifies that Verilog net strengths should be ignored when comparing two Verilog nets. (default) Related Tcl variable is PrefCompare(defaultIgnoreVerilogStrengths).
- **-internal**
(optional) Specifies comparison of internal (non-port) signals. Multiple modes can be specified in the same command.

- `-listwin <name>`
 (optional) Causes specified comparisons to be displayed in the specified List window. Related Tcl variable is `PrefCompare(defaultListWindow)`.
- `-maxsignal <n>`
 (optional) Specifies an upper limit for the total differences encountered on any one signal. When that limit is reached, ModelSim stops computing differences on that signal. Related Tcl variable is `PrefCompare(defaultMaxSignalErrors)`.
 <n> — Any positive integer where the default is 100.
- `-maxtotal <n>`
 (optional) Specifies an upper limit for the total differences encountered. When that limit is reached, ModelSim stops computing differences. Related Tcl variable is `PrefCompare(defaultMaxTotalErrors)`.
 <n> — Any positive integer where the default is 100.
- `-noaddwave`
 (optional) Specifies that new comparison objects are not added automatically to the Wave window. The default is to add comparison objects automatically. Related Tcl variable is `PrefCompare(defaultAddToWave)`.
- `-noignoreVlogStrengths`
 (optional) Specifies that Verilog net strengths should *not* be ignored when comparing two Verilog nets. Related Tcl variable is `PrefCompare(defaultIgnoreVerilogStrengths)`.
- `-notrack`
 (optional) Specifies that the waveform comparison should *not* track the current simulation. Related Tcl variable is `PrefCompare(defaultTrackLiveSim)`.
- `-out`
 (optional) Specifies comparison of ports of mode OUT. Multiple modes can be specified in the same command.
- `-ports`
 (optional) Specifies comparison of all ports. This switch has the same effect as specifying **-in**, **-out**, and **-inout** together.
- `-separator <string>`
 (optional) Used with the **-rebuild** option of the [compare add](#) command. When a bus has been broken into bits (bit blasted) by a synthesis tool, ModelSim expects a separator between the base bus name and the bit indication. This option identifies that separator. The default is an underscore (_). For example, the signal "mybus" might be broken down into "mybus_0", "mybus_1", etc. Related Tcl variable is `PrefCompare(defaultRebuildSeparator)`.

<string> — Specifies the character(s) to be used as a separator between the base bus name and the bit indication where the default is an underline (_).

- -tol <time>[<unit>]

(optional) Specifies the default maximum amount of time the test signal edge is allowed to trail or lead the reference edge in an asynchronous comparison.

You can specify different values for the leading and trailing tolerances using **-tolLead** and **-tolTrail**.

<time> — Any non-negative integer where the default is 0.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).

- -tolLead <time>[<unit>]

(optional) Specifies the default maximum amount of time the test signal edge is allowed to lead the reference edge in an asynchronous comparison. Related Tcl variables are PrefCompare(defaultLeadTolerance) and PrefCompare(defaultLeadUnits).

<time> — Any non-negative integer where the default is 0.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).

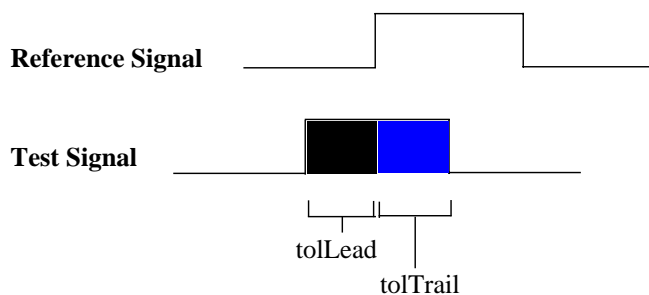
- -tolTrail <time>[<unit>]

(optional) Specifies the default maximum amount of time the test signal edge is allowed to trail the reference edge in an asynchronous comparison. Related Tcl variables are PrefCompare(defaultTrailTolerance) and PrefCompare(defaultTrailUnits).

<time> — Any non-negative integer where the default is 0.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).

Graphical representation of tolLead and tolTrail



- **-track**
(optional) Specifies that the waveform comparison should track the current simulation.
(default) The differences will be updated at the end of each **run** command, so if you want to see differences soon after they occur, use many relatively short run commands. Related Tcl variable is PrefCompare(defaultTrackLiveSim).
- **-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}**
(optional) Specifies how VHDL signal states in the reference dataset should match values in the test dataset. Values are specified in a colon-separated list of match values. For example:
-vhdlmatches {X=XUD:Z=ZD:1=1HD}
Default is:
{U=UWX- : X=UWXD : 0=0LD : 1=1HD : Z=ZD : W=UWXD : L=0LD : H=1HD : -=UX01ZWLHD}
Related Tcl variable is PrefCompare(defaultVHDLMatches).
- **-vlogmatches {<ref-logic-value>=<test-logic-value>:...}**
(optional) Specifies how Verilog signal states in the reference dataset should match values in the test dataset. Values are specified in a colon-separated list of match values. For example:
-vlogmatches {0=0:1=1:Z=Z}
Default is:
{0=0 : 1=1 : Z=Z : X=X}
Related Tcl variable is PrefCompare(defaultVLOGMatches).
- **-wavepane <n>**
(optional) Specifies the pane of the Wave window in which compare differences will be placed.
<n> — A positive integer corresponding to the pane of the Wave window. Wave window panes are numbered in sequence starting with 1 for the initial wave window.
- **-wavewin <name>**
(optional) Specifies the name of the Wave window in which compare differences will be viewed. Related Tcl variable is PrefCompare(defaultWaveWindow).
<wname> — The name of the window to which compare differences are to be added.

Examples

- Return the current value of all options.
compare options
- Set the maxtotal option to 2000 differences.
compare options -maxtotal 2000
- Return the current value of the maxtotal option.

compare options -maxtotal

- Set the option to ignore Verilog net strengths.

compare options -ignoreVlogStrengths

- Verilog X will now match X, Z, or 0.

compare options -vlogxmatches {0=0:1=1:Z=Z:X=XZ0}

- VHDL std_logic X will now match 'U', 'X', 'W', or 'D'.

compare options -vhdlmatches {X=UXWD}

- Set the leading tolerance for asynchronous comparisons to 300 picoseconds.

compare options -tolLead {300 ps}

- Set the trailing tolerance for asynchronous comparisons to 250 picoseconds.

compare options -tolTrail {250 ps}

Related Topics

- [compare add](#)
- [compare clock](#)
- [Waveform Compare](#)

compare reload

This command reloads comparison differences to allow their viewing without recomputation. Prior to invoking **compare reload**, you must open the relevant datasets with the same names that were used during the original comparison.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
compare reload <rulesFilename> <diffsFilename>
```

Arguments

- <rulesFilename>
(required) Specifies the name of the file that was previously saved using the **compare saverules** command. Must be the first argument to the **compare reload** command.
- <diffsFilename>
(required) Specifies the name of the file that was previously saved using the **compare savediffs** command.

Related Topics

- [compare add](#)
- [compare run](#)
- [compare savediffs](#)
- [compare saverules](#)
- [compare start](#)
- [Waveform Compare](#)

compare reset

This command clears the current compare differences, allowing another compare run command to be executed. Does not modify any of the compare options or any of the signals selected for comparison. This allows you to re-run the comparison with different options or with a modified signal list.

Syntax

```
compare reset
```

Arguments

- None

Related Topics

- [compare add](#)
- [compare run](#)
- [Waveform Compare](#)

compare run

This command runs the difference computation on the signals selected via a **compare add** command. Reports in the Transcript window the total number of errors found.

Syntax

```
compare run [<startTime>[<unit>]] [<endTime>[<unit>]]
```

Arguments

- `<startTime>[<unit>]`
(optional) Specifies when to start computing differences. Any positive integer, where the default is zero (0). Simulation resolution can be changed with the **-t** argument of the [vsim](#) command).
`<unit>` — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting `<unit>`. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If `<unit>` is specified, you must enclose `<startTime>` and `<unit>` within curly braces ({}).
- `<endTime>[<unit>]`
(optional) Specifies when to end computing differences. Any positive integer, where the default is the end of the dataset simulation run that ends earliest.
`<unit>` — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting `<unit>`. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If `<unit>` is specified, you must enclose `<endTime>` and `<unit>` within curly braces ({}).

Examples

- Compute differences over the entire time range.

```
compare run
```
- Compute differences from 5.3 nanoseconds to 57 milliseconds.

```
compare run {5.3 ns} {57 ms}
```

Related Topics

- [compare add](#)
- [compare end](#)
- [compare start](#)
- [Waveform Compare](#)

compare savediffs

This command saves the comparison results to a file for later reloading. To be able to reload the file, you must also save the comparison setup using the [compare saverules](#) command.

Syntax

```
compare savediffs <diffsFilename>
```

Arguments

- <diffsFilename>
(required) Specifies the name of the file to create.

Related Topics

- [compare add](#)
- [compare reload](#)
- [compare saverules](#)
- [Waveform Compare](#)

compare saverules

This command saves the comparison setup information (or "rules") to a file that can be re-executed later. The command saves compare options, clock definitions, and region and signal selections.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
compare saverules [-expand] <rulesFilename>
```

Arguments

- **-expand**
(optional) Expands groups specified by the **compare add** command to individual signals. If you added a region with the **compare add** command and then deleted signals from that region, you must use the **-expand** argument or the rules will not reflect the signal deletions.
- <rulesFilename>
(required) Specifies the name of the file to which you want to save the rules. To load the file at a later time, use the [compare reload](#) command. Must be specified as the final argument to the **compare saverules** command.

Related Topics

- [compare add](#)
- [compare reload](#)
- [compare savediffs](#)
- [Waveform Compare](#)

compare see

This command displays the specified comparison difference in the Wave window using whatever horizontal and vertical scrolling are necessary. The signal containing the specified difference will be highlighted, and the active cursor will be positioned at the starting time of the difference.

Syntax

```
compare see [-first] [-last] [-next] [-nextanno] [-previous] [-prevanno] [-wavepane <n>]  
           [-wavewin <name>]
```

Arguments

- -first
(optional) Shows the first difference, ordered by time. Performs the same action as the Find First Difference button in the Wave window.
- -last
(optional) Shows the last difference, ordered by time. Performs the same action as the Find Last Difference button in the Wave window.
- -next
(optional) Shows the next difference (in time) after the currently selected difference. Performs the same action as the Find Next Difference button in the Wave window.
- -nextanno
(optional) Shows the next annotated difference (in time) after the currently selected difference. Performs the same action as the Next Annotated Difference button in the Wave window.
- -previous
(optional) Shows the previous difference (in time) before the currently selected difference. Performs the same action as the Previous Difference button in the Wave window.
- -prevanno
(optional) Shows the previous annotated difference (in time) before the currently selected difference. Performs the same action as the Previous Annotated Difference button in the Wave window.
- -wavepane <n>
(optional) Specifies the pane of the Wave window in which the differences will be placed.

<n> — A positive integer corresponding to the pane of the Wave window. Wave window panes are numbered in sequence starting with 1 for the initial wave window.
- -wavewin <name>
(optional) Adds objects to a specified window. Used to specify a particular window when multiple instances of that window type exist.

<name> — The name of the window to which objects are to be added.

Examples

- Show the earliest difference (in time) in the default Wave window.

compare see -first

- Show the next difference (in time) in the default Wave window.

compare see -next

Related Topics

- [compare add](#)
- [compare run](#)
- [Waveform Compare](#)

compare start

This command begins a new dataset comparison. The datasets that you'll be comparing must already be open.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
compare start [-batch] [-maxsignal <n>] [-maxtotal <n>] [-refDelay <delay>[<unit>] [<unit>]]  
             [-testDelay <delay>[<unit>] [<unit>]] [<test_dataset>] <reference_dataset>
```

Arguments

- -batch
(optional) Specifies that comparisons will not be automatically inserted into the Wave window.
- -maxsignal <n>
(optional) Specifies an upper limit for the total differences encountered on any one signal. When that limit is reached, ModelSim stops computing differences on that signal. You can change the default using the [compare options](#) command or by editing the PrefCompare(defaultMaxSignalErrors) variable in the *pref.tcl* file.

<n> — Any integer where the default is 100.
- -maxtotal <n>
(optional) Specifies an upper limit for the total differences encountered. When that limit is reached, ModelSim stops computing differences. You can change the default using the [compare options](#) command or by editing the PrefCompare(defaultMaxTotalErrors) variable in the *pref.tcl* file.

<n> — Any integer where the default is 100.
- -refDelay <delay>[<unit>]
(optional) Delays the reference dataset relative to the test dataset. Delays are applied to signals specified with the [compare add](#) command. For each signal compared, a delayed virtual signal is created with "_d" appended to the signal name, and these are the signals viewed in the Wave window comparison objects. The delay is not applied to signals specified in compare "when" expressions.

<delay> — Any non-negative integer.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <delay> and <unit> within curly braces ({}).

- `-testDelay <delay>[<unit>]`

(optional) Delays the test dataset relative to the reference dataset. Delays are applied to signals specified with the [compare add](#) command. For each signal compared, a delayed virtual signal is created with "_d" appended to the signal name, and these are the signals viewed in the Wave window comparison objects. The delay is not applied to signals specified in compare "when" expressions.

`<delay>` — Any non-negative integer.

`<unit>` — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting `<unit>`. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If `<unit>` is specified, you must enclose `<delay>` and `<unit>` within curly braces (`{ }`).
- `<reference_dataset>`

(required) The dataset to be used as the comparison reference. Must be specified as the final argument to the **compare start** command unless you also specify the `<test_dataset>` argument.
- `<test_dataset>`

(optional) The dataset to be tested against the reference. If not specified, ModelSim uses the current simulation. The reference and test datasets may be the same.

Examples

- Begin a waveform comparison between a dataset named "gold" and the current simulation. Assumes the gold dataset was already opened.

```
compare start gold
```

- This command sequence opens two datasets and starts a comparison between the two using greater than default limits for total differences encountered.

```
dataset open gold_typ.wlf gold  
dataset open bad_typ.wlf test  
compare start -maxtotal 5000 -maxsignal 1000 gold test
```

Related Topics

- [compare add](#)
- [compare options](#)
- [compare stop](#)
- [Waveform Compare](#)

compare stop

This command is used internally by the **compare stop** button to suspend comparison computations in progress. If a [compare run](#) execution has returned to the VSIM prompt, **compare stop** has no effect. Under Unix, entering a Control-C character in the window that invoked ModelSim has the same effect as **compare stop**.

Syntax

```
compare stop
```

Arguments

- None

Related Topics

- [compare run](#)
- [compare start](#)
- [Waveform Compare](#)

compare update

This command is primarily used internally to update the comparison differences when comparing a live simulation against a *.wlf* file. The **compare update** command is called automatically at the completion of each simulation run if the [compare options -track](#) is in effect.

The user can also call **compare update** periodically during a long simulation run to cause difference computations to catch up with the simulation. This command does nothing if the **compare option -track** command was not in effect when the [compare run](#) command was executed.

Syntax

```
compare update
```

Arguments

- None

Related Topics

- [compare run](#)
- [Waveform Compare](#)

configure

The **configure** command invokes the List or Wave widget configure command for the current default List or Wave window.

To change the default window, use the [view](#) command.

Some arguments to this command are order-dependent. Please read through the arguments for further information.

Syntax

```
configure list | wave [-window <wname>] [<option> <value>]
```

---- List Window Arguments

```
[-delta [all | collapse | events | none]] [-gateduration [<duration_open>]]  
[-gateexpr [<expression>]] [-usegating [off | on]] [-strobeperiod [<period><unit>]]  
[-strobestart [<start_time><unit>]] [-usesignaltriggers [0 | 1]] [-usestrobe [0 | 1]]
```

---- Wave Window Arguments

```
[-childrowmargin [<pixels>]] [-cursorlockcolor [<color>]] [-gridauto [off | on]]  
[-gridcolor [<color>]][-griddelta [<pixels>]] [-gridoffset [<time><unit>]] [-gridperiod  
<time><unit>]]  
[-namecolwidth [<width>]] [-rowmargin [<pixels>]] [-signalnamewidth [<value>]]  
[-timecolor [<color>]] [-timeline [0 | 1]]  
[-timelineunits [fs | ps | ns | us | ms | sec | min | hr]] [-valuecolwidth [<width>]]  
[-vectorcolor [<color>]] [-waveselectcolor [<color>]] [-waveselectenable [0 | 1]]
```

Description

The command works in three modes:

- without options or values it returns a list of all attributes and their current values
- with just an option argument (without a value) it returns the current value of that attribute
- with one or more option-value pairs it changes the values of the specified attributes to the new values

The returned information has five fields for each attribute: the command-line switch, the Tk widget resource name, the Tk class name, the default value, and the current value.

Arguments

- list | wave

(required) Controls the widget to configure. Must be specified as the first argument to the **configure** command.

list — Specifies the List widget.

wave — Specifies the Wave widget.

- -window <wname>
(optional) Specifies the name of the List or Wave window to target for the **configure** command. (The **view** command allows you to create more than one List or Wave window). If no window is specified the default window is used; the default window is determined by the most recent invocation of the view command.
- <option> <value>
 - bg <color> — (optional) Specifies the window background color.
 - fg <color> — (optional) Specifies the window foreground color.
 - selectbackground <color> — (optional) Specifies the window background color when selected.
 - selectforeground <color> — (optional) Specifies the window foreground color when selected.
 - font — (optional) Specifies the font used in the widget.
 - height <pixels> — (optional) Specifies the height in pixels of each row. .

Arguments, List window only

- -delta [all | collapse | events | none]
(optional) Specifies how information is displayed in the delta column. To use **-delta**, **-usesignaltriggers** must be set to 1 (on).
 - all — Displays a new line for each time step on which objects change.
 - collapse — Displays the final value for each time step.
 - events — Displays an "event" column rather than a "delta" column and sorts List window data by event.
 - none — Turns off the display of the delta column.
- -gateduration [<duration_open>]
(optional) Extends gating beyond the back edge (the last list row in which the expression evaluates to true). The duration for gating to remain open beyond when **-gateexpr** (below) becomes false, expressed in x number of timescale units. The default value for normal synchronous gating is zero. If **-gateduration** is set to a non-zero value, a simulation value will be displayed after the gate expression becomes false (if you don't want the values displayed, set **-gateduration** to zero).
 - <duration_open> — Any non-negative integer where the default is 0 (values are not displayed).
- -gateexpr [<expression>]
(optional) Specifies the expression for trigger gating. (Use the **-usegating** argument to enable trigger gating.) The expression is evaluated when the List window would normally have displayed a row of data.

<expression> — An expression. See the [GUI_expression_format](#) for information on expression syntax.

- -usegating [off | on]

(optional) Enables triggers to be gated on or off by an overriding expression. (Use the **-gatexpr** argument to specify the expression.) Refer to “[Using Gating Expressions to Control Triggering](#)” for additional information on using gating with triggers.

off — (default) Triggers are gated off (a value of 0).

on — Triggers are gated on (a value of 1).

- -strobeperiod [<period>[<unit>]]

(optional) Specifies the period of the list strobe.

<period> — Any non-negative integer.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <delay> and <unit> within curly braces ({}).

- -strobestart [<start_time>[<unit>]]

(optional) Specifies the start time of the list strobe.

<start_time> — Any non-negative integer.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <delay> and <unit> within curly braces ({}).

- -usesignaltriggers [0 | 1]

(optional) Specifies whether or not signals are to be used as triggers.

0 — Signals are not used as triggers

1 — Signals are used as triggers

- -usestrobe [0 | 1]

(optional) Specifies whether or not a strobe is used as a trigger.

0 — Strobe is not used to trigger.

1 — Strobe is used to trigger.

Arguments, Wave window only

- -childrowmargin [<pixels>]

(optional) Specifies the distance in pixels between child signals. Related Tcl variable is PrefWave(childRowMargin).

<pixels> — Any non-negative integer where the default is 2.

- -cursorlockcolor [<color>]
(optional) Specifies the color of a locked cursor. Related Tcl variable is PrefWave(cursorLockColor).
<color> — Any Tcl color where the default is red.
- -gridauto [off | on]
(optional) Controls the grid period when in simulation time mode.
off — (default) user-specified grid period is used.
on — grid period is determined by the major tick marks in the time line.
- -gridcolor [<color>]
(optional) Specifies the background grid color. Related Tcl variable is PrefWave(gridColor).
<color> — Any color where the default is grey50.
- -griddelta [<pixels>]
(optional) Specifies the closest (in pixels) two grid lines can be drawn before intermediate lines will be removed. Related Tcl variable is PrefWave(gridDelta).
<pixels> — Any non-negative integer where the default is 40.
- -gridoffset [<time>[<unit>]]
(optional) Specifies the time (in user time units) of the first grid line. Related Tcl variable is PrefWave(gridOffset).
<time> — Any non-negative integer where the default is 0.
<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <delay> and <unit> within curly braces ({}).
- -gridperiod [<time>[<unit>]]
(optional) Specifies the time (in user time units) between subsequent grid lines. Related Tcl variable is PrefWave(gridPeriod).
<time> — Any non-negative integer where the default is 1.
<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <delay> and <unit> within curly braces ({}).
- -namecolwidth [<width>]
(optional) Specifies the width of the name column in pixels. Related Tcl variable is PrefWave(nameColWidth).
<width> — Any non-negative integer where the default is 150.

- **-rowmargin** [<pixels>]
(optional) Specifies the distance between top-level signals in pixels. Related Tcl variable is PrefWave(rowMargin).
 <pixels> — Any non-negative integer where the default is 4.
- **-signalnamewidth** [<value>]
(optional) Controls the number of hierarchical regions displayed as part of a signal name shown in the pathname pane. Related Tcl variable is PrefWave(SignalNameWidth). Can also be set with the WaveSignalNameWidth variable in the *modelsim.ini* file.
 <value> — Any non-negative integer where the default is 0 (display the full path. For example, 1 displays only the leaf path element, 2 displays the last two path elements, and so on.
- **-timecolor** [<color>]
(optional) Specifies the time axis color. Related Tcl variable is PrefWave(timeColor).
 <color> — Any color where the default is green.
- **-timeline** [0 | 1]
(optional) Specifies whether the horizontal axis displays simulation time or grid period count. Related Tcl variable is PrefWave(timeline).
 0 — (default) Simulation time is displayed.
 1 — Grid period count is displayed.
- **-timelineunits** [fs | ps | ns | us | ms | sec | min | hr]
(optional) Specifies units for timeline display. Does not affect the currently-defined simulation time.
 fs — femtosecond (10^{-15} seconds)
 ps — picosecond (10^{-12} seconds)
 ns — nanosecond (10^{-9} seconds) (default)
 us — microsecond (10^{-6} seconds)
 ms — millisecond (10^{-3} seconds)
 sec — second
 min — minute (60 seconds)
 hr — hour (3600 seconds)
- **-valuecolwidth** [<width>]
(optional) Specifies the width of the value column, in pixels. Related Tcl variable is PrefWave(valueColWidth).
 <width> — Any non-negative integer where the default is 100.

- `-vectorcolor [<color>]`
(optional) Specifies the vector waveform color. Default is #b3ffb3. Related Tcl variable is PrefWave(vectorColor).
`<color>` — Any color where the default is #b3ffb3.
- `-waveselectcolor [<color>]`
(optional) Specifies the background highlight color of a selected waveform. Related Tcl variable is PrefWave(waveSelectColor).
`<color>` — Any color where the default is grey30.
- `-waveselectenable [0 | 1]`
(optional) Specifies whether the waveform background highlights when an object is selected. Related Tcl variable is PrefWave(waveSelectEnabled).
0 — (default) Highlighting is disabled.
1 — Highlighting is enabled.

To get a more readable listing of all attributes and current values, use the [lecho](#) command, which pretty-prints a Tcl list.

There are more options than are listed here. See the output of a `configure list` or `configure wave` command for all options.

Examples

- Display the current value of the strobeposition attribute.
config list -strobeposition
- Set the period of the list strobe and turns it on.
config list -strobeposition {50 ns} -strobeposition 0 -usestrobe 1
- Set the wave vector color to blue.
config wave -vectorcolor blue
- Set the display in the current Wave window to show only the leaf path of each signal.
config wave -signalnamewidth 1

Related Topics

- [view](#)
- [Simulator GUI Preferences](#)

context

This command provides several operations on a context's name. The option you specify determines the operation.

For some of the commands there is a required argument "<name>". Please read through the arguments for further information.

Syntax

```
context dataset
context exists <name>
context fullpath <name>
context isInst <name>
context isNet <name>
context sProc <name>
context isVar <name>
context join <name>
context parent <name>
context path <name>
context split <name>
context tail <name>
type <name>
```

Arguments

- dataset
Returns the dataset name.
- exists <name>
Verifies that <name> exists in the design.

<name> — (required) Name of a context object or region. Does not have to be a valid object name.

Returns:
0 — <name> does not exist.
1 — <name> exists.
- fullpath <name>
Returns the full path (including the dataset prefix) of the specified <name>.

<name> — (required) Name of a context object or region.

- **isInst <name>**
Verifies whether <name> is an instance pathname or not.
<name> — (required) Name of a context object or region. Does not have to be a valid object name.
Returns:
 - 0 — <name> is not an instance pathname.
 - 1 — <name> is an instance pathname.
- **isNet <name>**
Verifies whether <name> is a Signal or Net pathname or not.
<name> — (required) Name of a context object or region. Does not have to be a valid object name.
Returns:
 - 0 — <name> is not a Signal or Net pathname.
 - 1 — <name> is a Signal or Net pathname.
- **isProc <name>**
Verifies whether <name> is a Process pathname or not.
<name> — (required) Name of a context object or region. Does not have to be a valid object name.
Returns:
 - 0 — <name> is not a Process pathname.
 - 1 — <name> is a Process pathname.
- **join <name> <name> ...**
Takes one or more <name> and combines them, using the correct path separator.
<name> — (required) Name of a context object or region. Does not have to be a valid object name.
- **parent <name>**
Returns the parent path of <name> by removing the tail (see context tail).
<name> — (required) Name of a context object or region.
- **path <name>**
Returns the pathname portion of <name>, removing the dataset name.
<name> — (required) Name of a context object or region.
- **split <name>**
Returns a list whose elements are the path components in <name>. The first element of the list will be the dataset name if one is present in the name, including the dataset separator. For example, context split /foo/bar/baz returns / foo bar baz.

context

<name> — (required) Name of a context object or region.

- tail <name>

Returns all of the characters in <name> after the last path separator. If <name> contains no separators then returns <name>. Any trailing path separator is discarded.

<name> — (required) Name of a context object or region.

- type <name>

Returns a string giving the acc type of <name>.

<name> — (required) Name of a context object or region.

coverage analyze

This command is used to display test oriented information based on a **vcover merge** operation. Some arguments for this command require that the merge to create the data was performed with the test-associated merge (performed without **-totals**).

This command is only available during post-simulation processing, when a UCDB file is opened with **vsim -viewcov**.

The coverage statistics displayed in the output of this command are calculated in accordance with the coverage aggregation algorithm shown in “[Calculation of Total Coverage](#)”.

Syntax

```
coverage analyze <miscellaneous coverage and filter args> <output mode args>
```

Miscellaneous Arguments

```
coverage analyze [-precision <int>] [-testextract <testspec>+] [-file <filename>] [-append]]
```

Filtering by coverage type

```
coverage analyze -code {b | c | e | f | s | t}...[-codeAll] [-assert] [-cvg] [-directive]
[-select instance]
```

Specifying coverage by path or context

```
coverage analyze
[[ -du <spec>] [-path <obj>]] | [-plansection <obj> [-section] | [-nosection]] [-recursive]
```

Filtering by query of results

```
coverage analyze
[-select attr <name> {-lt|-gt|-le|-ge|-eq|-ne|-[n]regexp} {<value>}]
[-select coverage {-lt|-gt|-le|-ge|-eq|-ne} <float>]
[-select tag {-eq|-ne|-[n]regexp} <string>]
[-select weight {-lt|-gt|-le|-ge|-eq|-ne} <int>]
[-anyselected | -allselected] [-aggregate preselected | postselected]
[-prunechildren | -noprunchildren]
```

Specifying output mode (mutually exclusive)

```
coverage analyze
[-coverage most | least | zero | nonzero]
| [-samples most | least | zero | nonzero]
| [-summary [hier|local]]
| [-totals]
```

Arguments

- -aggregate preselected | postselected
(optional: Only has an effect with the -select options (other than instance), can only be used with -plansection.)

Compute coverage for parent nodes before the selection of children, in other words based on *all* children (preselected)-- or after the selection, in other words based only on *selected* children (postselected).

- -anyselected | -allselected

(optional: Only has an effect with multiple -select options (does not work with -select instance)). These switches are mutually exclusive and only one may be given on each command line.) Use any select criteria (logical OR) or all select criteria (logical AND) to determine what is displayed in the report.

- -append

(optional) Appends to output file, if -file option was given.

- -assert

(optional) Specifies the command apply to assertion data. Assertions contribute to coverage based on "% non-vacuous passes".

- -code {b | c | e | f | s | t}...

(optional) Specifies the command apply to corresponding code coverage type: branch, condition, expression, statement, toggle, FSM.

- -codeAll

(optional) Specifies the command apply to all coverage types. Equivalent to -code bcestf.

- [-coverage most | least | zero | nonzero]

(optional) Reports which test yields most / least / zero / coverage for the specified parameters. This argument is on a per test basis. This argument requires that a test-associated merge (default unless -totals was specified) was performed.

- -cvg

(optional) Specifies the command apply to covergroup data.

- -directive

(optional) Specifies the command apply to directive data.

- -du <spec>

(optional, though either -du, -path, or -plansection must be specified.) Restricts analyze results to specified design units.

- -file <filename>

(optional) Write to output file.

- -nosection

(optional) Excludes section numbers (specified with -plansection) from the test information displayed. Allows for more concise output. Mutually exclusive with -section.

- `-path <obj>`
(optional, though either `-du`, `-path`, or `-plansection` must be specified.) Restricts coverage analysis results to design (non-testplan) paths matching the specified object. The `<obj>` can be used to specify a dataset other than the current dataset. (See [Object Name Syntax](#) for instructions on how to specify a dataset.) If no dataset is specified, the current dataset is used. Only one dataset name per command invocation may be used or an error will result. This switch can be combined with the `-du` switch to be du-relative.
- `-plansection <obj>`
(optional) Restricts the analyze results to the given testplan path. By default, the query applies to all testplan scopes in the database. It also applies to all linked coverage, instance, or design unit scopes. Either `-du`, `-path`, or `-plansection` must be specified.
- `-precision <int>`
(optional) Precision for output.
- `-prunechildren` | `-noprunechildren`
(optional) The `-prunechildren` switch prunes children from nodes surviving a selection (after `-select` is applied) from the report. By default (`-noprunechildren`), the children of nodes are included in the coverage analyze report: the report includes all children of coverage items or test plan sections that survive a `-select` filter. This is useful in cases where you want to subdivide a test plan report by category or area of responsibility.
- `-recursive`
(optional) Specifies that the analyze is applied recursively. The default is for the query to be restricted to the single object or objects specified in the command.
- `[-samples most | least | zero | nonzero]`
(optional) Reports which test has the optionally computed sample count, meeting the specified criteria. The specified criteria are: most samples, least samples, no samples, or any samples. This argument is on a per test basis. Optional. Using this argument implies the use of `-cvg`, since only covergroups and covergroup instances have sample counts. This argument is not compatible with a merge performed with `-totals`.
- `-section`
(optional) Includes section numbers (specified with `-plansection`) in the test information displayed. This is the default. Mutually exclusive with `-nosection`.
- `-select attr <name> {-lt|-gt|-le|-ge|-eq|-ne|-[n]regexp} {<value>}`
(optional) Displays user attributes in the report. The `<name>` of the user-defined attribute is a string and must be given as the second argument. The following rules apply to name/value pairs:
 - `-lt|-gt|-le` | `-ge` require a numerical argument.

-lt is "less than", -gt is "greater than", -ge is "greater than or equal to", -le is "less than or equal to", -eq is "equal to", and -ne is "not equal to", -regex is "regular expression match", -nregex is "regular expression does not match".

- -regex and -nregex require a string argument.
- The -regex option uses Tcl regular expression matching (as in the memory window); this offers alternative and more sophisticated matching. One crucial difference is that "match anything starting with 'a'" is expressed differently in the two semantics: as "-eq a*" and "-re ^a.*".
- For -eq or -ne, the argument is held as a string and will match int or float or double attribute values as appropriate (for example, "2" could match any type attribute).
- <value> can be a float, integer or string. In cases where an attribute is of a type incompatible with the comparison value, a single warning is issued.
- -select coverage {-lt|-gt|-le|-ge|-eq|-ne} <float>
(optional) Includes the coverage of the items in the test information displayed.
- -select instance
(optional) Includes design instances only in the report: Verilog module, program, and interface instances, VHDL architectures, and package instances.
- -select tag {-eq|-ne|-n]regex} <string>
(optional) Reports the tag used to associate the specified object with the test plan, or to associate the object with the other objects. The tag is assigned with the [coverage tag](#) command. Tags can be used for primitive (non-hierarchical) grouping of coverage in the database. Tags are used for test plan to coverage linking as well.
 - tag can only be used with -eq, -ne, or -n]regex and a string. <string> can be a pattern with wildcards.

-eq is "equal to", and -ne is "not equal to", -regex is "regular expression match", -nregex is "regular expression does not match".
 - The -regex option uses Tcl regular expression matching (as in the memory window); this offers alternative and more sophisticated matching. One crucial difference is that "match anything starting with 'a'" is expressed differently in the two semantics: as "-eq a*" and "-re ^a.*".
- select -unlinked
Obsolete command. Use "[coverage unlinked](#)" instead.
- -select weight {-lt|-gt|-le|-ge|-eq|-ne} <int>
(optional) Includes the weights of the coverage objects in the test information displayed. You set the weight for coverage objects using the "[coverage weight \(-du|-path|-plan\)](#)" command.

- -lt is "less than", -gt is "greater than", -ge is "greater than or equal to", -le is "less than or equal to", -eq is "equal to", and -ne is "not equal to", -regexp is "regular expression match", -nregexp is "regular expression does not match". Optional.
- -summary [hier|local]
(optional) Generates a summary (stats) style report for the given path(s). The default argument to -summary is "hier".
 - "hier" generates statistics recursively including sub-instances of the design hierarchy, if used with -path. If used with -plansection, "hier" is test plan section recursive.
 - "local" generates statistics for the local instance only.

This argument can not be used with **-du**. If used with **-path**, the **-select instance** argument is applied automatically, which restricts the command to design instance scopes; the summary report is most useful (i.e., shows multiple kinds of coverage) if used with design instances. For other types of coverage, e.g., covergroup and FSM coverage hierarchy, the -totals report is much more concise.

The summary includes Assertion Passes, Failures, Successes and Attempts. At simulation, time if "-assertdebug" is present, Assertion Passes are counted. Assertion Successes are those assertions that never failed and passed at least once. In the absence of "-assertdebug," Assertion Passes are not counted, and Assertion Successes are those assertions that never failed. Assertion Attempts are the sum of all evaluated assertions.

- -testextract <testspec>+
(optional) For a vcover merge, this reconstructs a report for the specified test or tests. <testspec> is a list of TESTNAME attribute names for all tests in the UCDB. Merge must have been performed without -totals.
- -totals
(optional) Generates a hierarchical report of coverage totals — as a single number — for test plan sections, based on children nodes and linked coverage items and instances or design units (those sharing a tag with the testplan scope.). Optional. Could also be used for HDL design unit or instance scopes. It calculates the average of all different types of coverage found, i.e. an average of results obtained with **-summary**. For instances or design units, the total coverage is an average of all types. The "coverage" is the total coverage number computed for a coverage object:
 - Weighted average of different kinds of coverage found within a design instance or design unit.
 - Design units only "contain" code coverage, not functional coverage or assertions.
 - The global per-type weights set with "[coverage weight](#)" can be used to affect the weighted average, but these weights apply globally.
 - Coverage result is for a particular kind of coverage object: covergroups, FSM, cover directive, toggle, branch, etc.

- Non-vacuous pass result (100% if any, 0% if none) for an assertion.

Hierarchically computed weighted average of children of a testplan section, can be computed pre or post selection with -aggregate preselected|postselected.

Examples

- Generate a hierarchical report of all testplan scopes, including linked scopes:

```
vsim> coverage analyze -plansection / -totals -r
```

```
Total Coverage Report
Sec#      Testplan Section / Coverage Link      Coverage      Goal Weight
-----
-         /testplan                               56.18% 100.00%    1
1         Top                                    56.18% 100.00%    1
1.1       Bits
          /top/child1/cvg_bits_vs_clock       83.33% 100.00%    2
          /top/child1/cvg_bits_vs_bits       89.06% 100.00%    1
1.2       Arithmetic
          /top/child2/cvg_arith               27.11% 100.00%    1
```

- Various examples of test queries:

Is there a test which has zero coverage for the named testplan?

```
vsim> coverage analyze -plansection /test1.0/test1.1 -coverage zero
test0
test4
```

Show the test which has least coverage for the named testplan. Show summary.

```
vsim> coverage analyze -plansection /test1.0/test1.2 -coverage least
-summary
test13:
Coverage Summary BY INSTANCES ...
```

Show the test which has most coverage for the named testplan. Show total only.

```
vsim> coverage analyze -plansection /test1.0/test1.3 -coverage most
-total
test27: 99%
```

Filter by cover directives only:

```
vsim> coverage analyze -plansection /test1.0/test1.3 -coverage most
-total -dir test27: 90%
```

Ask the question based on a particular covergroup:

```
vsim> coverage analyze -path /top/i/cvg1 -coverage most -total
test23: 85%
```

Display anything not completely covered in a test plan:

```
coverage analyze -plansection / -r -select cover -lt 100
```

Display coverage holes with weight greater than 1:

```
coverage analyze -plansection / -r -select cover -eq 0 -weight -gt 1
```

Display any coverage tagged "essential" that is less than 100% covered:

```
coverage analyze -plansection / -r -select cover -lt 100 -select tag
-eq essential
```

Because test plan sections numbers can be used as tags for coverage linking, this can show Section 4 of a test plan:

```
coverage analyze -plansection / -r -select tag -eq 4*
```

Match a user-defined attribute "myattr" whose value starts with "a", or one which does not start with "a":

```
coverage analyze -plansection / -r -select attr myattr -re "^a.*"
```

```
coverage analyze -plansection / -r -select attr myattr -nre "^a.*"
```

See also

[Code Coverage](#), [“Calculation of Total Coverage”](#), [Coverage and Verification Management in the UCDB](#), [“Verification Browser Window”](#), [“Verification Tracker Window”](#), [“Viewing Test Data in the Tracker Window”](#), [coverage attribute](#), [coverage goal](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage unlinked](#), [coverage weight](#), [vcover merge](#), [vcover ranktest](#), [vcover report](#), [vcover stats](#)

coverage attribute

The **coverage attribute** command is used to display or set attributes in the currently loaded database on the following types of attributes:

- **Test Attributes** — attributes for each test attribute record (one record is created for each simulation that is saved). These attributes are name value pairs that represent testcase information. Refer to the section "[Predefined Attribute Data](#)" for complete list of these attributes.
- **UCDB Attributes** — attached globally to the UCDB file, read or written with "coverage attribute -ucdb". Unlike test attributes, these are merged together during a vcover merge. In the current system, the only attributes created by ModelSim are those related to the test-associated merge. However, you can create attributes for your own use, accessible through this CLI or the UCDB API.
- **Object Attributes** — attached to particular objects stored in the UCDB (ex. design units, design instance scopes, a particular covergroup, or a particular cover directive). Some attributes for different kinds of objects are created by ModelSim, but you can create or read any attribute in the CLI or the UCDB API.

This command can be used both during simulation and with "vsim -viewcov", though in simulation it can only be used for test attributes (the single test attribute record that exists in simulation).

To apply filters (-select instance, -assert, -code, etc.):

1. Match paths first, with recursion (if specified).
2. Specify paths to be "thrown out" (those not matching the filter).

Syntax

To display or set test attributes

```
coverage attribute [-test <testname>] [-seed <str>] [-command <str>] [-comment <str>]
[-compulsory [0|1]] [-delete] [-tcl] [-concise] [[-name <str> -value <str>]...]
```

To display or set UCDB attributes

```
coverage attribute [-ucdb] [-tcl] [-concise] [[-name <str> -value <str>]...]
```

To display or set object attributes

```
coverage attribute [-match <str> | -path <obj> | -plansection <obj>]
[-du <duname>] [-select instance]
[-assert] [-code {b | c | e | f | s | t}...] [-codeAll] [-cvlg] [-directive]
[-tcl] [-concise] [[-name <str> -value <str>]...]
```

Arguments

- -assert
(optional) Specifies this command applies to assertion data.

- `-code {b | c | e | f | s | t}...`
(optional) Specifies this command applies to corresponding code coverage types: branch, condition, expression, statement, toggle, FSM.
- `-codeAll`
(optional) Specifies this command applies to all coverage types. Equivalent to `-code bcestf`.
- `-command <str>`
(optional) Command to run the test: script command line, "knob settings", etc.
- `-comment <str>`
(optional) Comment on the testcase.
- `-compulsory [0|1]`
(optional) Indicates test is compulsory. By default, it is not compulsory (0).
- `-concise`
(optional) Print attribute values only, do not print other information.
- `-cvg`
(optional) Specifies the command apply to covergroup data.
- `-delete`
(optional) Delete specified name attributes.
- `-directive`
(optional) Specifies the command apply to directive data.
- `-du <duname>`
(optional) Apply to a design unit, e.g., "lib.primary(secondary)" secondary for VHDL only.
- `-match <str>`
(optional) Recursively matches the given pattern against the specified coverage types in the entire instance tree. If `-duname` is specified, it matches against the specified coverage types in the design unit. Wildcards are allowed.
- `-name <str>`
(optional) Sets an attribute name. Used to add user defined attributes to either a test, a UCDB, or a particular object in the UCDB. Multiple `-name` arguments are allowed.
- `-path <obj>`
(optional) Apply to a path in the UCDB. The `<obj>` can be used to specify a dataset other than the current dataset. (See Object Name Syntax for instructions on how to specify a dataset.) If no dataset is specified, the current dataset is used. Only one dataset name per command invocation may be used or an error will result. Wildcards are acceptable. Relative path can be used in conjunction with the `-du` switch.

- `-plansection <obj>`
(optional) Apply to a testplan section in the UCDB, as specified by `<obj>`. Wildcards are acceptable. Relative path can be used in conjunction with `-du`.
- `-seed <str>`
(optional) Random seed of the test run.
- `-select instance`
(optional) Specifies the command applies to HDL instance scopes (VHDL architectures, interface instances, etc.).
- `-tcl`
(optional) Prints attribute information in a Tcl format.
- `-test <testname>`
(Required when used with `vsim -viewcov`; optional otherwise.) Specifies a test object for attributes.
- `-ucdb`
Required when used with `vsim -viewcov`. Optional otherwise. Specifies global UCDB object for attributes.
- `-value <str>`
Required when used with `vsim -viewcov`. Optional otherwise. Used only with `-name`. Value of attribute associated with `-name`. Multiple `-value` arguments are allowed.

Example

- Show all test records in a UCDB that has been loaded into coverage view mode:

```
coverage attribute -test *
```

See also

[“Verification Browser Window”](#), [“Understanding Stored Test Data in the UCDB”](#), [coverage exclude](#), [coverage goal](#), [coverage analyze](#), [coverage ranktest](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage testnames](#), [coverage weight](#), [coverage unlinked](#), [vcover attribute](#), [vcover merge](#), [vcover ranktest](#), [vcover stats](#)

coverage clear

The **coverage clear** command clears specified types of coverage data from the coverage database.

When entered at the simulation prompt (simulation mode), performing coverage clear on an instance affects the code coverage data of the associated design unit. The reverse is also true, that if you perform "coverage clear" on a design unit, the associated instances of that design unit are also cleared.

However, when issued at the vsim prompt with the vsim -viewcov command (batch or post-processing modes), coverage clear does not synchronize code coverage data between instances and associated design units. So, clearing an instance has no effect on code coverage data for associated design units. Conversely, clearing a design unit has no effect on related instances.

Syntax

```
coverage clear [-code {b | c | e | f | s | t |}]... [-codeAll]
               [-assert] [-cvg] [-directive]
               [-du <du_name> | -instance <pathname>] [-path <obj>+] [-match <string>] [-recursive]
```

Arguments

- -assert
(optional) Clears assertion data only.
- -code {b | c | e | f | s | t |}...
(optional) Clears code coverage data for coverage type: b=branch coverage; c=condition coverage; e=expression coverage; s=statement coverage; t=toggle; f=Finite State Machine coverage. More than one of the coverage types may be specified with a single argument.
- -codeAll
(optional) Specifies the command apply to all coverage types. Equivalent to -code bcestf.
- -cvg
(optional) Clears covergroup data only.
- -directive
(optional) Clears cover directive data only.
- -du <du_name>
(optional) Specifies design unit to clear of specified types of coverage data. To specify all design units in the current dataset, specify <du_name> as "*".
- -instance <pathname>
(optional) Clears the specified coverage data for the specified instances; -instance can be specified multiple times.

- **-match <string>**
(optional) Recursively matches the given pattern against the specified coverage types in the entire instance tree. If **-duname** is specified, it matches against the specified coverage types in the design unit. Wildcards are allowed. This switch is mutually exclusive with **-recursive**.
- **-path <obj>+**
(optional) Specifies that the subtrees being cleared are rooted at the specified design node. Multiple objects may be specified. The **<obj>** can be used to specify a dataset other than the current dataset. (See “[Object Name Syntax](#)” for instructions on how to specify a dataset.) If no dataset is specified, the current dataset is used. Only one dataset name per command invocation may be used or an error will result. This switch applies to a sub-hierarchy.
- **-recursive**
(optional) Specifies that the command is applied recursively. Optional. The default is for the query to be restricted to the single object or objects specified in the command. This switch is mutually exclusive with **-match**.

Example

- `coverage clear`
Clears all coverage data from the current simulation database (UCDB).
- `coverage clear -cvg -directive`
Clears data for all covergroups and covergroup directives.
- `coverage clear -path /top/a/*`
Clears coverage data from all */top/a*.

See also

[Code Coverage](#), [coverage attribute](#), [coverage exclude](#), [coverage ranktest](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage analyze](#)

coverage edit

The **coverage edit** command opens a coverage dataset (.ucdb) to edit the contents. Used only in Coverage View mode (vsim -viewcov). Use this command to create UCDBs with specific subsets of the overall coverage.

Syntax

To prune away objects from the UCDB:

```
coverage edit -keeponly | -delete
    { -path <path> [-recursive] |
      -plansection <path> [-recursive] |
      -du <path> [-recursive] |
      <coverage_types> }...
```

where <coverage_types> are:

```
( [-code {b | c | e | f | s | t | }...] [-codeAll] [-cvg] [-directive] [-assert] )
```

Arguments

- -assert
(optional) Deletes or keeps all assertions.
- -code {b | c | e | f | s | t | }...
(optional) Clears code coverage data for coverage type: b=branch coverage; c=condition coverage; e=expression coverage; s=statement coverage; t=toggle; f=Finite State Machine coverage. More than one of the coverage types may be specified with a single argument.
- -codeAll
(optional) Specifies the command apply to all coverage types. Equivalent to -code bcestf.
- -cvg
Deletes or keeps covergroup data only. Optional.
- -delete
(optional) Deletes specified UCDB subset or coverage types. Mutually exclusive with **-keeponly**.
- -directive
(optional) Deletes or keeps cover directive data only.
- -du <path>
(optional) Deletes or keeps the specified design unit. Mutually exclusive with -plan and <coverage types>.
- -keeponly
(optional) Keeps only given UCDB subset or coverage types. Mutually exclusive with **-delete**.

- `-path <path>`
(optional) Deletes or keeps the specified design or testbench hierarchy. Mutually exclusive with `-plan`, and all `<coverage types>`.
- `-plansection <path>`
(optional) Deletes or keeps the specified test plan section. Mutually exclusive with `-path` and `<coverage types>`.
- `-recursive`
(optional) Specifies that exclusions apply recursively into subsopes. Optional. If omitted, the exclusions are limited to the current scope.

Examples

- Discard statement coverage from the entire database (the `-path / -r` is assumed):

```
coverage edit -delete -code s
```

- Keep only coverage from a specified test plan section:

```
coverage edit -keeponly -plansection /myplan/1.2
```

- Keep only coverage from a specified design unit:

```
coverage edit -keeponly -du /top/a
```

See also

[“Coverage View Mode and the UCDB”](#), [“Coverage and Verification Management in the UCDB”](#), [“Verification Browser Window”](#), [coverage analyze](#), [coverage attribute](#), [coverage exclude](#), [coverage ranktest](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage weight](#), [vsim -viewcov option](#)

coverage exclude

The **coverage exclude** command allows you to exclude the following from coverage statistics:

- specific code coverage items (statement, branch, expression or condition)
- specific code coverage types
- all code in specified source file(s)
- lines within a source file
- specific items on a line within a source file
- rows within a condition or expression truth table
- code inside specific design units or instances
- transitions or states within a Finite State Machine
- toggle nodes
- assertions
- cover directives

This command and its arguments can be issued during simulation or in Coverage View (post-process) mode. Refer to “[Coverage Exclusions](#)” for more details.

File based exclusions cannot be cleared by scope. For example, an exclusion that was set using `-srcfile` cannot be cleared later using `-scope`.

Syntax

For file-based, line-based, or wholesale exclusions of coverage:

```
coverage exclude
{ -srcfile <source_file_list> [-pragma] |
  -du <du_name> [-srcfile <source_file_list>] [-pragma] |
  -scope <scope_path> [-srcfile <source_file_list>] [-recursive] }
[-linerange [<ln>] ... [<ln>-<ln>] ...] [-item {<bces>} [<int> | <int-int>]+] [-allfalse]
[-dataset <name>] [-code {b | c | e | f | s | t}...] [-clear]
```

To exclude expression or condition rows:

```
coverage exclude
{ -srcfile <source_file_list> [-pragma] |
  -du <du_name> [-srcfile <source_file_list>] [-pragma] |
  -scope <scope_path> [-srcfile <source_file_list>] }
[-condrow <ln> [<rn>] ... [<rn>-<rn>] ...]
[-exprrow <ln> [<rn>] ... [<rn>-<rn>] ...]
[-item {<bces>} [<int> | <int-int>]+]
[-fecondrow <ln> [<rn>] ... [<rn>-<rn>] ...]
```

```
[-fecexprrow <ln> [<rn>] ... [<rn>-<rn>] ...]  
[-dataset <name>] [-clear]
```

To exclude functional coverage (assertions, cover directives, or covergroups) by file, by instance, or by design unit:

```
coverage exclude  
{ -assertpath <assert_list> | -dirpath <directive_list> | -cvgpath <cvg_list> }  
[-srcfile <source_file_list> | -du <du_name>] [-clear]
```

To exclude FSM states or transitions:

```
coverage exclude  
{ -du <du_name> [-pragma] |  
-scope <scope_path> }  
{ -ftrans <state_var_name> [<transition_name>] ... |  
-fstate <state_var_name> [<state_name>] ... }  
[-dataset <name>] [-clear]
```

To exclude an entire state machine from coverage:

- if auto exclusions are enabled:

```
coverage exclude -fstate <state_var_name>
```

- if auto exclusions are not enabled:

```
coverage exclude -fstate <state_var_name> -ftrans <state_var_name>
```

To exclude toggle coverage:

```
coverage exclude  
-togglenode <node_path_list> [-du <du_name> | -scope <scope_path> [-recursive]]  
[-dataset <name>] [-in] [-out] [-inout] [-internal] [-ports] [-clear] [-pragma]
```

Arguments

- -allfalse

(optional) Modifies branch exclusion algorithm by applying exclusions to the false path of a branch when the branch does not have an explicit "else". This argument applies to branch coverage only. Branch coverage (on by default) must be turned on for this argument to take effect. The line number(s) specified with the -linerange argument, if used, must include the line on which the if-branch appears. For more information about allfalse and if-else branches, see "[Branch Coverage](#)".

- -assertpath <assert_list>

(optional) Excludes the assertion with the specified path from coverage.

- -code {b | c | e | f | s | t}...

(optional) Excludes coverage objects of the specified type from the specified dataset.

b=branch coverage; c=condition coverage; e=expression coverage; f=Finite State Machine coverage; s=statement coverage; t=toggle coverage (either regular or extended). If -code is

specified without any modifier, all possible coverage types are excluded. More than one coverage can be specified with each `-code` argument. If `-item` is used, **-code f** or **t** is not valid.

- `-clear`
(optional) Removes exclusions from dataset. Add exclusions if `-clear` is not specified.
- `-condrow <ln> [<rn>] ... [<rn>-<rn>] ...`
(optional) Specifies condition truth table row(s) *<rn>* in the specified line *<ln>* to be excluded from coverage. Multiple rows, or ranges of rows, separated by spaces, are allowed. If no row number is specified, all rows are excluded.
- `-cvgpath <cvg_list>`
(optional) Excludes the covergroup with the specified path from coverage.
- `-dataset <name>`
(optional) Specifies dataset into which exclusions are to be applied. Only one dataset name per command invocation may be used or an error will result. If not specified, the current dataset is assumed ("sim" is the default when running interactively). All specified objects, such as scopes, design units, or variable names, must be present in the named dataset. (See Object Name Syntax for instructions on how to specify a dataset.)
- `-dirpath <directive_list>`
(optional) Excludes the cover directive with the specified path from coverage.
- `-du <du_name>`
(required only for du-based exclusions) Specifies design unit to be excluded. Multiple `-du` specifications are allowed. Mutually exclusive with `-scope`, which is the default. The *<du_name>* must include the path to the design unit being excluded. To specify all design units in the current dataset, specify *<du_name>* as "*" (e.g. coverage exclude -du *). You cannot use `-du` with `-srcfile` or `-linerange` when *<du_name>* is "*".
- `-exprrow <ln> [<rn>] ... [<rn>-<rn>] ...`
(optional) Specifies expression truth table row(s) *<rn>* in the specified line *<ln>* to be excluded from coverage. Multiple rows, or ranges of rows, separated by spaces, are allowed. If no row number is specified, all rows are excluded.
- `-feccondrow <ln> [<rn>] ... [<rn>-<rn>] ...`
(optional) Excludes specified row in focused expression coverage (FEC) condition coverage with a specified line number from the report. Multiple rows, or ranges of rows, separated by spaces, are allowed. If no row number is specified, all rows are excluded.
- `-fecexprrow <ln> [<rn>] ... [<rn>-<rn>] ...`
(optional) Excludes specified row in focused expression coverage (FEC) expression coverage with a specified line number from the report. Multiple rows, or ranges of rows, separated by spaces, are allowed. If no row number is specified, all rows are excluded.

- `-fstate <state_var_name> [<state_name>] ...`

(optional) Specifies the Finite State Machine state or states to be excluded from coverage for the specified FSM, specified with `<state_var_name>`. Multiple states, separated by white space, are allowed. If no state name is specified, all states are excluded. By default, when a state is excluded, all transitions to and from the state are excluded. This behavior is called "auto exclusion". To explicitly control auto exclusion, set the **vsim** argument `-autoexclusionsdisable` to `fsm` or `none`. To change the default behavior of the tool, set the variable **AutoExclusionsDisable** in the `modelsim.ini` file.
- `-frans <state_var_name> [<transition_name>] ...`

(optional) Specifies the transition states to be excluded for the specified FSM (`state_var_name`). `<transition_name>` is "`<state_name>-><state_name>`". Multiple transitions, separated by white space, are allowed. If no transition is specified, all transitions are excluded. If whitespace is present within the transition, it must be surrounded by curly braces.
- `-in`

(optional) Excludes the specified toggle nodes of mode IN. This argument is valid only when `-togglenode` is specified.
- `-inout`

(optional) Excludes the specified toggle nodes of mode INOUT. This argument is valid only when `-togglenode` is specified.
- `-internal`

(optional) Excludes the specified toggle nodes of internal (non-port) objects. This argument is valid only when `-togglenode` is specified.
- `-item {<bces>} [<int> | <int-int>]+`

(optional) Excludes specified coverage item(s) on a line of source code from database. The `-item` argument can only be applied to coverage exclude command entries for the line number specified with `-linerange`, `-condrow`, or `-exprrow`. `<bces>` is required and is used to specify one or more of the coverage types to exclude: branch, condition, expression, and/or statement. Items are numbered in left to right order within a line, regardless of hierarchy, from 1 upward. Only one `-item` argument allowed with each coverage exclude command. This argument may not be used with the **-code tf** argument.
- `-linerange [<ln>] ... [<ln>-<ln>] ...`

(optional) Specifies the line number(s) and/or range of line numbers to be excluded from code coverage in the design source file `-srcfile <source_file>`. Multiple lines and line ranges are permitted, separated by whitespace.

 - When `-linerange` is not specified, all objects on all lines of the specified design unit, scope, or source file are excluded. This is referred to as a "wholesale exclusion".

- -srcfile is required for -linerange unless -du or -scope is used, and only one source file is used to implement the du or scope.
- If -srcfile is used together with -du/-scope, and -linerange is in effect, it is possible for -linerange to specify lines other than lines used to implement the -du or -scope. Such lines are ignored.
- -out
(optional) Excludes the specified toggle nodes of mode OUT. This argument is valid only when -togglenode is specified.
- -ports
(optional) Excludes the specified toggle nodes of mode IN, OUT, or INOUT. This argument is valid only when -togglenode is specified.
- -pragma
(optional) Adds or clears pragma and user exclusions. Operates with file-based exclusions (-du and/or -srcfile) for all coverage types, including toggle exclusions (-togglenode). If the -pragma argument is specified, both user and pragma exclusions are applied. If the option is not specified, only user exclusions are applied.
- -recursive
(optional) Used with -scope only. Specifies that exclusions apply recursively into subscopes. If omitted, the exclusions are limited to the current scope.
- -scope <scope_path>
(required only for instance-based exclusions) Specifies the scope to be excluded. Multiple -scope specifications are allowed. Mutually exclusive with the -du argument. -scope is the default if neither argument is used. To recursively exclude scopes, use with -r.
- -srcfile <source_file_list>
(required only for file-based exclusions) Specifies source file to be excluded. Multiple file specifications are allowed, separated by white space.
- -togglenode <node_path_list>
(optional) Specifies the named nodes for toggle exclusion. Multiple nodes separated by spaces are allowed. Wildcards and part-selects are accepted, using the following format:
 - * — matches any substring within a level of hierarchy
 - toggle[<int>:<int>] — matches an integer index within the range
 - toggle(<int|*> to <int|*>) — matches an integer index within the range
 - toggle(<int|*> downto <int|*>) — matches an integer index within the range

By default, specified toggle nodes to be excluded are relative to the current scope. If used with -du, specified toggle nodes excluded are relative to the design unit.

Examples

- Recursively exclude branch coverage from instance */top/dut*.

```
coverage exclude -scope /top/dut -r -code b
```
- Exclude statement, else branch, expression and condition coverage from line 10 to 20 in file *project1.vhd*.

```
coverage exclude -srcfile project1.vhd -linerange 10-20
```
- Exclude statement, branch, expression, and condition coverage from instance */top/dut* in dataset *tt* from line 102 through 110 and line 200 through 250 in the source file *project1.vhd*.

```
coverage exclude -scope /top/dut -dataset tt -srcfile  
project1.vhd -linerange 102-110 200-250
```
- Remove statement, branch, expression, condition, and fsm exclusions from the source file *project1.vhd*.

```
coverage exclude -clear -srcfile project1.vhd
```
- Add rows 2 through 4 from the condition truth table on line 115 to the code coverage exclusions for source file *project1.vhd*.

```
coverage exclude -srcfile project1.vhd -condrow 115 2-4
```
- Add all rows from the expression truth table on line 220 to the code coverage exclusions for source file *project1.vhd*.

```
coverage exclude -srcfile project1.vhd -exprrow 220
```

or

```
coverage exclude -srcfile project1.vhd -linerange 220 -code e
```
- Exclude transitions S1->S2 and S2->S0 for FSM state in instance */top/dut/fsm1*.

```
coverage exclude -scope /top/dut/fsm1 -ftrans state S1->S2 S2->S0
```
- Exclude state S1 for FSM state in the design unit "fsm". If auto exclusions are on, all transitions to and from S1 are also excluded.

```
coverage exclude -du fsm -fstate state S1
```
- Remove user and pragma exclusions for all toggle coverage. This is equivalent to 'toggle enable -all'.

```
coverage exclude -du * -code t -clear -pragma
```
- Exclude all toggle coverage (equivalent to 'toggle disable -all')

```
coverage exclude -du * -code t -pragma
```

- Exclude toggle nodes a, b, and c in instance */top/dut*.

```
coverage exclude -togglenode a b c -scope /top/dut
```
- Recursively exclude all input toggle nodes in instance */top/dut*.

```
coverage exclude -togglenode * -scope /top/dut -in -r
```
- Exclude covergroup at */top/tb/pci_cg*:

```
coverage exclude -cvgpath /top/tb/pci_cg
```

What NOT to do: Illegal Examples

```
coverage exclude -srcfile project1.vhd -code s -allfalse
```

- -allfalse has no effect because branch coverage is not specified.

```
coverage exclude -srcfile project1.vhd -linerange 10-20 -code ft
```

- There is no file name and line number associated with FSM and toggle coverage.

```
coverage exclude -scope /top/dut -srcfile project1.vhd -linerange 10-20 -r
```

- -r does not work with -srcfile or -linerange

```
coverage exclude -du * -srcfile project1.vhd -linerange 10-20
```

- '-du *' does not work with -srcfile or -linerange

```
coverage exclude -scope /top/dut -srcfile project1.vhd -line 10-20 -pragma
```

- -pragma does not work with -scope

See also

[“Code Coverage”](#), [“Coverage Exclusions”](#), [Coverage and Verification Management in the UCDB](#), [“Verification Browser Window”](#), [coverage analyze](#), [coverage report](#), [coverage save](#), [“Toggle Coverage”](#), [toggle add](#), [toggle enable](#), [toggle disable](#)

coverage goal

The **coverage goal** command sets the value of UCDB-wide goals for different coverage types, or goals for specific objects in the database.

For more information regarding type coverage statistics gathered, see “[Calculation of Total Coverage](#)”.

Syntax

```
coverage goal [-cvp] [-bydu] [-byinstance] [-type] [-fstate] [-ftrans] [-fail] [-pass] [-vpass]
  [-disabled] [-attempted] [-active] [-precision <int>] [<float percentage>]
  [-du <du_name> | -path <path> | -plansection <section_name>] [-assert] [-cvg] [-code {[b]
  [c] [e] [f] [s] [t]}] [-codeAll] [-directive]
```

Arguments

- -active
Assertion directive active, per instance. Optional.
- -assert
(optional) Specifies the command for assertion data only.
- -bydu
(optional) Modifier used to set per-du (code coverage only)
- -byinstance
(optional) Modifier used to set a per-instance goal (code coverage and covergroup).
- -code {[b] [c] [e] [f] [s] [t]}
(optional) Sets goal for code coverage data for coverage type: b=branch coverage; c=condition coverage; e=expression coverage; s=statement coverage; t=toggle; f=Finite State Machine coverage. More than one coverage type can be (and at least one type must be) specified with each -code argument.
- -codeAll
(optional) Specifies the command for all coverage types. Equivalent to -code bcestf.
- -cvg
(optional) Specifies the command for covergroup data only.
- -cvp
(optional) Select coverpoint per-instance coverage.
- -directive
(optional) Specifies the command for directive data only.

- **-du <du_name>**
(optional) Sets the goal for a given design unit. Mutually exclusive with **-path** and **-plansection**. Cannot be combined with any other arguments besides **-precision** or **<float percentage>**.
- **-fstate**
(optional) Selects FSM state coverage.
- **-ftrans**
(optional) Selects FSM transition coverage.
- Assertion directive arguments are as follows.
 - fail**
(optional) Assertion directive failures, per instance.
 - pass**
(optional) Assertion directive passes, per instance.
 - vpass**
(optional) Assertion directive vacuous passes, per instance.
 - disabled**
(optional) Assertion directive disabled, per instance.
 - attempted**
(optional) Assertion directive attempted, per instance.

Note



These have no effect in the simulator because these do not contribute to coverage calculations. They are still stored in the database, however.

- **<float percentage>**
(required in order to set goals: prints goal(s) if left unspecified) Value for goal or goal(s) between 0 and 100. Required in order to set goals: prints goal(s) if left unspecified.
- **-path <path>**
(optional) Sets the goal for a given coverage/design object. Mutually exclusive with **-plansection** and **-du**. **<path>** can be used to specify a dataset other than the current dataset. If no dataset is specified, the current dataset is used. Only one dataset name per command invocation may be used or an error will result. Cannot be combined with any other arguments besides **-precision** or **<float percentage>**.
- **-plansection <section_name>**
(optional) Sets the goal for a given test plan item. Mutually exclusive with **-path** and **-du**. Cannot be combined with any other arguments besides **-precision** or **<float percentage>**.

- -precision <int>
(optional) Precision for goal percentage. Default is 1 decimal place.
- -type
(optional) Modifier used to set covergroup type coverage.

See also

[Code Coverage, Coverage and Verification Management in the UCDB](#)", "[Verification Browser Window](#)", [coverage analyze](#), [coverage attribute](#), [coverage exclude](#), [coverage ranktest](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage weight](#)

coverage open

The **coverage open** command opens UCDB datasets for viewing in the GUI in Coverage View mode. Datasets can be closed once open using dataset close.

This command is equivalent to the command `vsim -viewcov`.

Syntax

```
coverage open <filename> [<logicalname>]
```

Arguments

- **<filename>**
(required) Specifies the <filename>.ucdb to open in Coverage View mode. At least one UCDB is required.
- **<logicalname>**
(optional) Specifies the logical name for the UCDB dataset. This is a prefix that will identify the dataset in the current session. By default the dataset prefix will be the name of the specified UCDB file.

Examples

- Open the dataset file *last.ucdb* and assigns it the logical name *test*.

```
coverage open last.ucdb test
```

See also

[“Coverage View Mode and the UCDB”](#), ["Coverage and Verification Management in the UCDB"](#), [“Verification Browser Window”](#), [“Verification Tracker Window”](#), [coverage analyze](#), [coverage attribute](#), [coverage exclude](#), [coverage ranktest](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage weight](#), [dataset close](#), [vsim -viewcov option](#)

coverage ranktest

The coverage ranktest command ranks coverage data contained in the current Coverage View dataset (whether loaded with vsim -viewcov or in batch), according to each individual test's contribution to cumulative coverage. The dataset should be from a merged UCDB, and have been created without the **vcover merge -totals** argument.

The output of this command consists of what is written to *stdout*, as well as two lists of tests (contributing tests in *ranktest.contrib* and non-contributing tests in *ranktest.noncontrib*). The order of the tests written to stdout or a specified file (with -log) are listed in the order shown in [Table 2-3](#):

Table 2-3. Order and Type of Ranked Tests

Contributing, compulsory	Mandatory tests, tests which need to be run regardless of achieved coverage	Sorted by total coverage %
Contributing, noncompulsory	Tests providing coverage not provided by any previous test.	Sorted by total coverage %
Non-contributing	Redundant tests, providing no incremental coverage.	Not sorted

To rank on a specific coverage item, design unit, or test plan section within the hierarchy of your design, use -path, -du or -plansection.

Syntax

```
coverage ranktest
  [-log <filename>]
  [<coverage_type>]
  [-nocompulsoryordering | -compulsorysorting | -compulsoryranking]
  [-goal [<coverage_type>] <percentage>]]
  [-weight <coverage_type> <integer>]
  [-maxcpu <real_num_in_seconds>] [-maxtests <int>]
  [-metric {aggregate | total}]
  [-fewest | -cputime | -simtime]
  [-path <path> | -du <du_name> | -plansection <path>]
  [-precision <int_num>]
  [-quiet | -concise | -verbose]
  [-rankfile <filename>]

<coverage_type> =
  [-assertion] [-assertfailure] [-code {b | c | e | f | s | t}...] [-codeAll] [-cvg] [-directive]
```


Arguments

- -assertion

(optional) Specifies ranktest for assertion data. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.

Traditionally, assertions are a measure of correctness and not a measure of coverage. When viewed as a coverage item, their definitions are as follows:

- Percent coverage assertion passes (-assertion): Of all assertions what percent were evaluated to completion and had at least one non-vacuous pass.
- Percent coverage assertion failures (-assertfailure): Of all assertions what percent were evaluated to completion and had at least one failure. This is THE ONLY NEGATIVE COVERAGE VALUE in the sense that 0% coverage is desirable and 100% coverage is very undesirable. As such, it is highly recommended that -assertfailure only be used in a mutually exclusive fashion with the other coverage values.

It is possible for any given assertion to have both passes and failures. It is also possible for any given assertion to not ever be evaluated to completion. As such, often the sum of assertion passes and assertion failures will not be 100%.

- -assertfailure

(optional) Specifies ranktest for failed assertion data only. See "[-assertion](#)" for more information. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.

- -code {b | c | e | f | s | t}...

(optional) Specifies ranktest for corresponding code coverage type only: branch, condition, expression, statement, toggle, FSM. More than one coverage type may be specified with each -code argument (example: "-code bcest"). Optional. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.

- -codeAll

(optional) Specifies ranktest for all coverage types. Equivalent to -code bcestf. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.

- [-nocompulsoryordering](#) | [-compulsorysorting](#) | [-compulsoryranking](#)

(optional) Specifies how the command ranks compulsory tests, when present.

-nocompulsoryordering — (default) Compulsory tests are listed in a random order, with no ranking with respect to each other. Compulsory tests are not displayed in the detailed ranktest listing. In the GUI, the accumulated coverage data and incremental coverage data for compulsory tests is displayed as a minus sign (-). This option has the best performance of the three options.

-compulsorysorting — Ranks compulsory tests in the order of descending 'total coverage' with respect to each other. Displays compulsory tests in the detailed ranktest listing according to this order. In the GUI, the accumulated coverage data

and incremental coverage data for compulsory tests is calculated and displayed. This option has performance second to the `-nocompulsoryordering` option.

`-compulsoryranking` — Ranks compulsory tests fully with respect to each other.

Displays compulsory tests in the detailed ranktest listing according to this order. In the GUI, the accumulated coverage data and incremental coverage data for compulsory tests will be calculated and displayed. This option can have the worst performance of the three options.

- `-concise`

(optional) Specifies the output is created with minimum additional I/O. This is the default output. Mutually exclusive with `-quiet` and `-verbose`.

- `-cputime`

(optional) Specifies that the tests be ranked by minimum CPU time. Mutually exclusive with the `-fewest` and `-simtime` arguments.

- `-cvg`

(optional) Specifies ranktest for covergroup data only. This argument is not compatible with `-metric total`: if both are explicitly set, ranking is halted and an error is issued.

- `-directive`

(optional) Specifies ranktest for directive data only. This argument is not compatible with `-metric total`: if both are explicitly set, ranking is halted and an error is issued.

- `-du <du_name>`

(optional) Restricts ranking to a specified design unit. Mutually exclusive with `-path` and `-plansection`.

- `-fewest`

(optional) Specifies that the tests be ranked by fewest number of tests. Mutually exclusive with the `-cputime` and `-simtime` arguments. Default.

- `-goal [<coverage_type>] <percentage>`

(optional) Specifies that the tests be ranked to achieve this coverage goal. This argument is not compatible with `-metric total`: if both are explicitly set, ranking is halted and an error is issued.

The `<coverage_type>` specifies that the goal is applied only to the specified type of coverage; otherwise, the goal is applied to all types of coverage. Valid `<coverage_type>`s are:

`-assertfailure`, `-code {b|c|e|f|s|t}...`, `-codeAll`, `-cvg`, or `-directive`.

`<percentage>` is an integer value. Default value for `<percentage>` is 100.

The `-goal` argument can be specified multiple times, as shown in this example:

```
coverage ranktest -goal -assertion 90 -goal -code bcest 95
```

- -log <filename>
(optional) Specifies the file for outputting ranked results.
- -maxcpu <real_num_in_seconds>
(optional) Monitors the accumulated CPU time of the ranked tests. Specifies the maximum CPU time to be allowed. If the specified number of seconds is exceeded, the ranking process is stopped. The default value is -1.0 (no limit).
- -maxtests <int>
(optional) Specifies threshold for the maximum number of tests to be ranked. When this threshold is exceeded, the ranking operation is terminated.
- -metric {aggregate | total}
(optional) Indicates the kind of metric used for ranking. Arguments:
total — (Default, unless any of the following arguments are used: **-totals**, **-goal**, **-weight**, **-assertion**, **-assertfailures**, **-cvg**, **-directive**, **-codeAll**, or **-code**)
Ranking metric used produces values consistent with the totals for (verification) test plan sections obtained with [coverage analyze](#).
aggregate —
Ranking metric used produces aggregate values based on each individual coverage type: values are *not* likely to be consistent with totals produced with [coverage analyze](#).
Each coverage type can be selected or not. Each coverage type can be given an individual weight and goal. These individual numbers are then combined and normalized to yield an aggregate metric that is unrelated to the number given by the total coverage.



Important: When the **-metric aggregate** argument is used, the resulting metric number will not “match” any other total coverage number produced by other verification tools (i.e. coverage analyze). This is important because when you use any of the arguments (**-totals**, **-goal**, **-weight**, **-assertion**, **-assertfailures**, **-cvg**, **-directive**, **-codeAll**, or **-code**) with ranktest command, the aggregate metric is the default.

- -path <path>
Restricts ranking to design paths (non-testplan) matching the specified <path>. Optional. This argument can be used to rank on specific coverage items, or sub-trees. Mutually exclusive with **-du** and **-plansection**.
- -plansection <path>
Restricts ranking to the specified testplan node. Optional. Mutually exclusive with **-du** and **-path**.

- **-precision <int_num>**
(optional) Specifies the decimal point precision for output. <int_num> is an integer value. The default value is 2.
- **-quiet**
(optional) Creates the ranktest output without any additional I/O. Default creates ranktest with minimal I/O (-concise). Mutually exclusive with -concise and -verbose.
- **-rankfile <filename>**
(optional) Specifies the name (including path) for a ranktest file, created by default by this command creates. Default filename created is *ranktest.rank*. Can be specified with the [vcover stats](#) command to redisplay the results of this ranking. This file can also be used to repopulate the Browser with ranktest information.
- **-simtime**
(optional) Specifies that the tests be ranked by minimum simulation time. Mutually exclusive to the **-cputime** and **-fewest** arguments.
- **-verbose**
(optional) Specifies the output created with full I/O. Mutually exclusive with **-quiet** and **-concise**. Default is -concise.
- **-weight <coverage_type> <integer>**
(optional) Used when selecting next ranking candidate. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.
<coverage_type> are: [-assertion](#), [-assertfailure](#), [-directive](#), [-cvg](#), [-code {b | c | e | f | s | t}](#)..., and [-codeAll](#).

The -weight argument can be specified multiple times, as shown in this example:

```
coverage ranktest -weight -assertion 10 -weight -code bcest 5
```

See also

[“Code Coverage”](#), ["Coverage and Verification Management in the UCDB"](#), [“Verification Browser Window”](#), [“Verification Tracker Window”](#), [coverage goal](#), [coverage tag](#), [coverage unlinked](#), [coverage weight](#), [vcover merge](#), [vcover ranktest](#), [vcover stats](#)

coverage report

The **coverage report** command produces textual output of coverage statistics or exclusions. By default, the command prints results to the Transcript window, and returns an empty string. You can use the `-file` argument to save the output to a file.

You can choose from a number of report output options using the arguments listed below. You can access the coverage report functionality from the GUI through right-clicking in the Structure or Files windows and select **Code Coverage > Coverage Reports** from the popup context menu; or, **Tools > Code Coverage > Report**.

By default, the command returns results from the current scope. To specify a certain path for the report, you can use the `-instance` argument, or specify the specific cover directive or covergroup, such as:

- `coverage report -instance <path>`
- `coverage report <path_to_cover_directive/covergroup>`



Tip: A report response of "No match" indicates that the report was empty. For example, "coverage report -du foo" where there is no design unit "foo" will result in "No match."

The command orders output on a by file basis unless you specify the **-byinstance** or **-bydu** argument.

To produce reports offline (i.e., without a simulation loaded), use the **vcover report** command.

Syntax

```
coverage report [<coverage_arguments>]
```

Global Arguments - Usable with any other arguments

```
coverage report [-above <percent> | -below <percent>] [-binrhs]
  [-details [-dumptables] [-fecanalysis] [-metricanalysis]] [-file <filename> [-append]]
  [-memory] [-noexcludedhits] [-nozeroweights] [-precision <int>] [-recursive [-depth <n>]]
  [-showambiguity] [-testextract <test_name_or_pattern>] [-xml] [-zeros]
```

Create HTML output from a UCDB

```
coverage report [-html [-code [bcesf[t|x]]] [-assert] [-cvg] [-directive] [-verbose] [-nosource] [-
  noframes] [-nodetails] [-summary] [-htmldir <outdir>] [-threshL <val>] [-threshH <val>]
  <input_ucdb>]
```

Filtering Arguments - Selects one or more coverage types to appear in the report

```
coverage report [-assert] [-code {[b] [c] [e] [f] [s] [t]}] [-codeAll] [-cvg] [-directive]
  [-lang sva | psl | vhdl] [-severity info | note | warning | error | failure | fatal] [-testattr]
```

For each directive instance, the report includes by default the full instance path, the coverage count or percentage, the design unit, the source file name, and the source line number.

Code Coverage - Has no effect on assertion or functional coverage

```
coverage report [-bydu] [-byfile] [-byinstance] [-coverenhanced] [-totals] [-noannotate]
  [-library <libname>] [-du <du_name>] [-package <pkgname>]
  [-setdefault [byfile | byinstance | bydu]] [-source <filename>]
  [-instance <path>] [-recursive [-depth <n>]]
```

Exclusion-specific Coverage Arguments

```
coverage report [-excluded [-pragma | -user]] [-code {[b] [c] [e] [f] [s] [t]}]
  [-instance <path>]
  [-file <filename> [-append]]
```

Toggle-specific Coverage Arguments

```
coverage report [-verbose] [-all]
```

Toggle coverage statistics are relevant only when reporting on instances or design units and are not produced on a per file basis. Toggle data is summed for all instances, and is reported by port or local name in the design unit, rather than by the connected signal. If you want toggle coverage statistics, you must specify either the **-byinstance**, **-bydu**, **-instance <path>**, or **-du <du_name>** arguments. If you do not use those arguments, or you use the **-source <filename>** argument, toggle coverage statistics are excluded even if you specify **-code t**. To get an itemized list of the signals, the **-details** argument is also required.

Assertion- specific Arguments

```
coverage report [-assert] [-lang sva | psl | vhdl] [-severity info | note | warning | error | failure |
  fatal] [-unattemptedimmed]
```

Cover directive-specific Arguments for SystemVerilog and PSL

```
coverage report [-config]
```

Covergroup-specific Arguments

```
coverage report [-nocvpcrossbinsummary] [-nocrossbinsummary]
  [-noignorebins] [-option] [-covered] [-usecnpm] [-samples <spec>]
```

Arguments

- **-above <percent>**
(optional) Specifies that only objects with coverage values above this percentage be included in the output. <percent> is the coverage of coverpoints and crosses, not covergroups.
- **-all**
(optional) When reporting toggles, creates a report that lists both toggled and untoggled signals. Counts of all enumeration values are reported. Not a valid option when reporting on a functional coverage database.
- **-append**
(optional) Appends the current coverage statistics to the named output file (-file <filename>).

- **-assert**
(optional) Reports only assertion coverage data.
- **-below <percent>**
(optional) Specifies that only objects with coverage values below this percentage be included in the output. <percent> is the coverage of coverpoints and crosses, not covergroups. See also **-above**.
- **-binrhs**
(optional) Specifies that for covergroups, a column is included in the report that displays the RHS for covergroup bins. The RHS is a sampled value that cause the bin to increment.
- **-bydu**
(optional) Reports coverage statistics by design unit/module. The simulator will iterate through all design units in the design and report coverage data for each. Each design unit report will be the sum of all instances of that design unit and will be sorted by design unit name. Can be used with the **-recursive [-depth <n>]** argument to report on all design units contained within the specified design unit. Can be made the default with the **-setdefault bydu** argument. You can also report coverage data for a specific design unit by using the **-du <du_name>** argument.
- **-byfile**
(optional) Writes out a coverage summary for each source file in the design. This is the default report generated. A report generated with **-byfile** does not contain toggle information.
- **-byinstance**
(optional) Writes out a coverage summary for all instances and packages. Can be replace the default **(-byfile)** with the **-setdefault byinstance** argument.
- **-code {[b] [c] [e] [f] [s] [t]}**
(optional) Specifies which code coverage statistics to include in the report. If **-code** is specified, the specification of one or more coverage types is required. By default, the report includes statistics for all categories you enabled at compile time. More than one coverage type can be specified with the **-code** argument.

Not a valid option when reporting on functional coverage data.

The coverage types allowed are as follows:

- b — Include branch statistics.
- c — Include condition statistics.
- e — Include expression statistics.
- f — Include finite state machine statistics.
- s — Include statement statistics.

t — Include toggle statistics.

To report extended toggle coverage, ensure that you have compiled (vlog/vcom) with the -code x argument, then use coverage report with -code t.

- -codeAll
(optional) Specifies the command apply to all coverage types. Equivalent to -code bcestf.
- -concurrent | -immediate
(optional) Only concurrent assertions are selected when the -concurrent switch is used. Only immediate assertions are selected when the -immediate switch is used. These are mutually exclusive options. If neither switch is specified, both concurrent and immediate assertions are selected.
- -config
(optional) Specifies that the current configuration of each cover directive be included in the report.
- -covered
(optional) Displays covered bins only. This argument is used only for covergroups in coverage reports and the GUI. If any covergroup item survives the covergroup filters applied, its ancestors are displayed in the report and the GUI.
- -coverenhanced
(optional) Enables functionality which may change the appearance or content of coverage metrics. You can find a detailed list of these changes in the release notes by searching for the string "coverenhanced".
- -cvg
(optional) Reports only covergroup coverage data.
- -details [-dumptables] [-fecanalysis] [-metricanalysis]
(optional) Includes details associated with each coverage item in the output (both UDP and FEC). By default, details are not provided.
 - dumptables — forces printing of condition and expression truth tables even though fully covered.
 - fecanalysis — reports which input patterns can be applied to the inputs to increment the expression/condition hit counts.
 - metricanalysis — prints sum-of-product and basic sub-condition heuristic metrics from UDP expression/condition view. It reports hit counts for all rows in UPD table. To improve coverage numbers, find rows with 0 hits and exercise the inputs accordingly. See [“Condition and Expression Coverage”](#) for more information on metrics.
- -directive
(optional) Reports only cover directive coverage data.

- `-du <du_name>`
(optional) Reports coverage statistics for the specified design unit. `<du_name>` is `<library name>.<primary>(<secondary>)`, where the library name is optional, and secondary name is required only for VHDL. If there are parameterized instances, all are considered to match the specified design unit.
- `-excluded [-pragma | -user]`
(optional) Includes details on the exclusions in the specified coverage database input file. The output is structured in Tcl command format (DO file).
By default, this option includes both user exclusions and source code pragma exclusions, unless you specify **-user** or **-pragma**.
 - `-pragma` — When used with the **-excluded** argument, writes out *only* lines currently being excluded by pragmas. Optional.
 - `-user` — When used with the **-excluded** argument, writes out files and lines currently being excluded by the **coverage exclude** command. Optional.
- `-file <filename>`
(optional) Specifies a file name for the report. Default is to write the report to the Transcript window. Environment variables may be used in the pathname.
- `-html [-code [bcesf[t|x]] [-assert] [-cvg] [-directive] [-verbose] [-nosource] [-noframes] [-nodetails] [-summary] [-htmldir <outdir>] [-threshL <val>] [-threshH <val>] <input_ucdb>`
(optional) Generates an HTML coverage report on coverage data from a given UCDB file. You can use the **-verbose** option with **-html** to enable logging output for each file generated. The **-html** arguments listed below are not compatible with any other vcover report arguments, with the exception of `-binrhs`, `-code`, `-assert`, `-cvg`, and `-directive`.
 - `<input_ucdb>` — Specifies input UCDB file. Required, and only one is allowed.
 - `[-code [bcesf[t|x]] [-assert] [-cvg] [-directive]` — Specifies types of coverage to include in the HTML coverage report. (A `-covergroup` switch exists which is synonymous to `-cvg` and may be used in its place.) If none of these four switches is defined, all coverage types are included in the report.
 - `-verbose` — Prints out the files that are generated by the HTML report generator. Optional.
 - `-nosource` — Avoids generation of the annotated source. Optional. This argument is used if you have no source code, or if you don't want the annotated source to be generated. Note that this prevents you from accessing source code related data from inside the generated HTML report.
 - `-noframes` — Avoids generation of JavaScript-based tree for designs with a large number of design scopes. The report comes up as a single frame containing the top-level summary page and an HTML-only design scope index page is available as a link from the top-level page.

- nodetails — Omits coverage detail pages, saving time and disk space during report generation for very large designs.
- summary — Includes only the top summary page, the testplan summary page, and the list of tests run in the generated report.
- htmldir <outdir> — Specifies the name of output directory for resulting UCDB (default: "covhtmlreport"). Optional. Whether you specify an output directory or the default is used, any file or directory of that name is completely removed prior to report generation to prevent possible stale data.
- threshL <%> -threshH <val> — Specifies % of coverage at which colored cells change from red to yellow. Optional.
- threshH <%> — Specifies % of coverage at which colored cells change from yellow to green. Optional.

The default output filename is *index.html* in the default directory, *covhtmlreport*.

- -instance <path>
(optional) Writes out the source file summary coverage data for the specified instance. The <path> can be used to specify a dataset other than the current dataset. (See Object Name Syntax for instructions on how to specify a dataset.) If no dataset is specified, the current dataset is used. Only one dataset name per command invocation may be used or an error will result.
- -lang sva | psl | vhdl
(optional) Specifies assertions of a specific language (SVA, PSL, or VHDL). You may specify multiple languages by using this option multiple times. If -lang is not specified, all three languages are selected.
- -library <libname>
(optional) Only needs to be used when you have packages of the same name in different libraries.
- -memory
(optional) Reports a coarse-grain analysis of capacity data for the following SystemVerilog constructs:
 - Classes
 - Queues, dynamic arrays, and associative arrays (QDAS)
 - Assertion and cover directives
 - Covergroups
 - Solver (calls to randomize())

When combined with -cvg and -details, this command reports the detailed memory usage of covergroup. These include the current persistent memory, current transient memory, peak transient memory, and peak time of the following:

- Per covergroup type
- Per coverpoint and cross in the type
- Per covergroup instance (if applicable)
- Per coverpoint and cross in the instance (if applicable).
- -noannotate
(optional) Removes source code from the output report. Valid for code coverage only. Not applicable with **-xml** argument.
- -nocrossbinsummary
(optional) Removes cross bins for each coverpoint and cross from the report output.
- -nocvpcrossbinsummary
(optional) Removes covered bins for each coverpoint and cross from the report output.
- -noexcludedhits
(optional) By default, if any excluded items are hit during simulation an “E-hit” notification is displayed in the text report. If “E-hit” is present, it overrides the display of “E” for an exclusion or “EA” for an autoexclusion. To disable display of “E-hit” notifications, use the **-noexcludedhits** option. This has the effect of making all “E” and “EA” notifications uniformly visible, along with exclusion reasons (if available).
- -noignorebins
(optional) Removes covergroup ignore bins from the report output.
- -nozeroweights
(optional) Removes covergroups, coverpoints, and crosses having zero weight from the report output.
- -option
(optional) Includes all covergroup **option** and **type_option** values in the report. Unless your covergroup has the "option.per_instance" set to true, only the **type_option** is included by default. Only applicable to covergroup reports created with the **-details** argument. Optional.
- -package <pkgname>
(optional) Prints a report on the specified VHDL package body. Needs to be of the form <lib>.<pkg>. This argument is equivalent to **-du**.
- -precision <int>
(optional) Sets the decimal precision for printing functional coverage information. Valid values are from 0 to 6 and default value is 1 (one).
- -recursive [-depth <n>]
(optional) Reports on the instance specified with **-instance** and every included instance, recursively. Can also be used with **-details** and **-totals** but *cannot* be used with **-zeros**.

-depth <n>

Used with the **-recursive** argument, it specifies the maximum recursive depth. A depth of 1 is the same as no recursion at all. Optional.

- **-samples <spec>**

(optional) Filters the sample count for covergroup types. The sample count is optionally calculated using [SVCovergroupSampleInfo](#), set in the *modelsim.ini* file. This argument filters to include covergroups whose sample count matches the criteria given <spec>. It is a filter for covergroup types only, not instances. If covergroups are found without a sample count, a warning is issued. <spec> is one of the following four mutually exclusive options:

[number] — sample count is equal to [number]

-le [number] — sample count is less than or equal to number

-ge [number] — sample count is greater than or equal to number

[lower]-[upper] — (no white space allowed) sample count is in the range lower-upper. There is only 1 -sample option allowed per invocation.

- **-setdefault [byfile | byinstance | bydu]**

(optional) Sets the coverage report default mode for the current invocation of ModelSim. Report modes are by file (default), by instance, and by design unit.

- **-severity info | note | warning | error | failure | fatal**

(optional) Specifies the assertion severity level. When -severity is specified only assertions with the same or higher severity are selected. If not specified, assertions of all severities are selected.

- **-showambiguity**

(optional) When used, coverage report displays both minimum and maximum counts for any conflicting toggle data in a UCDB that results from a combined merge ([vcover merge](#) command performed with -combine).

- **-source <filename>**

(optional) Writes a summary of statement coverage data for a specific source file. Environment variables may be used in the pathname.

- **-testattr**

(optional) Displays test attributes in the report.

- **-testextract <test_name_or_pattern>**

(optional) Displays test specific results in the report. Used to combine results from multiple tests. The <test_name_or_pattern> is the test or pattern to extract. Multiple -testextract arguments can be applied in same command. This argument is compatible with reports generated in plain text and XML formats only, HTML reports are not supported. When using this argument, a header line appears at the top of the report listing test name(s) used to generate the report. Also, the word “hit” appears in place of the count number. UCDB files

store only the aggregated coverage counts from all tests, and test-specific numbers can not be reproduced.

- -totals

(optional) Writes out a total summary of the specified instance, recursively. Useful for tracking changes. Without this argument, the report writes out an instance summary for each of the instances. The report prints only one summary if -totals option is used. Also, when the -totals argument is specified, the alias nodes are not counted.

The total summary includes Assertion Passes, Failures, Successes and Attempts. At simulation, time if "-assertdebug" is present, Assertion Passes are counted. Assertion Successes are those assertions that never failed and passed at least once. In the absence of "-assertdebug," Assertion Passes are not counted, and Assertion Successes are those assertions that never failed. Assertion Attempts are the sum of all evaluated assertions.

- -unattempteddimmed

(optional) Causes any unexecuted immediate assertions to be considered in the coverage calculations for Total Coverage that are displayed in the coverage report. By default, any unexecuted immediate assertions are not included in the report.

- -usecnpm

(optional) Specifies that the value of [SVCrossNumPrintMissingDefault](#) is used in report. By default, all cross bins are displayed in the report.

- -verbose

(optional) Prints a report listing all the integer values and their counts an integer toggle encounters during the run. List will include the number of active assertion threads (Active Count) and number of active root threads (Peak Active Count) that have occurred up to the current time.

- -xml

(optional) Outputs report in XML format. A report created with -xml does not contain source file lines (calls **-noannotate** implicitly). This implicitly sets the **-details** argument. Refer to "[Coverage Reports](#)" for more information.

- -zeros

(optional) Writes out a file-based summary of lines, including file names and line numbers, that have not been executed (zero hits), annotates the source code, and supports the **-source** and **-instance** options. Cannot be used in tandem with the **-recursive** argument.

For covergroups, this argument applies to coverpoint and cross bins only.

For a detailed report that includes line numbers, use: **coverage report -zeros -details**.

Examples

- Write a top-level summary of the number of files, statements, branches, hits, and signal toggles to *myreport.txt*.

```
coverage report -totals -file myreport.txt
```

- Write detailed branch, condition, and statement statistics, without associated source code, to the transcript window.

```
coverage report -details -noannotate -code bcs
```

- Write a summary of code coverage for all instances to the Transcript window.

```
coverage report -byinstance
```

- Write code coverage details of all instances in the design to *myreport.txt*. The **-details** argument reports coverage statistics for each statement, branch, condition and expression.

```
coverage report -details -byinstance -file myreport.txt
```

- Write code coverage details of one specific instance to the Transcript window.

```
coverage report -details -instance /top/p
```

- Write toggle data from the test *clyde40ns*, listed by design unit, including both toggled and untoggled signals.

```
coverage report -details -testextract clyde40ns -bydu -code t -all
```

- Write both pragma and user-based exclusions to the transcript window as follows:

```
coverage report -excluded

# coverage report -excluded
#   src/delta/delta.vhd
#     693-696
#     711-806
#   src/delta/micro.v
#     110-124
#   src/delta/pre.v
#     216-217
#   src/delta/testdel.vhd
#     1178-1274
#   src/delta/tx.vhd
#     148-149
```

- Write both pragma and user-based exclusions to the transcript window in TCL format as follows:

```
# coverage report -excluded
# coverage exclude -add src/delta/delta.vhd 693-696 711-806
# coverage exclude -add src/delta/micro.v 110-124
# coverage exclude -add src/delta/pre.v 216-217
# coverage exclude -add src/delta/testdel.vhd 1178-1274
# coverage exclude -add src/delta/tx.vhd 148-149
```

- Write a summary of coverage by source file for coverage less than or equal to 90%.

```
coverage report -below 90 -file myreport.txt
```

- Write a list of statements with zero coverage to *myzerocov.txt*.

```
coverage report -zeros -byinstance -file myzerocov.txt
```

See also

[“Code Coverage”](#), [“Generating HTML Coverage Reports”](#), [coverage save](#), [vcover report](#), [coverage analyze](#), [coverage attribute](#), [coverage ranktest](#), [coverage tag](#), [coverage goal](#), [coverage weight](#), [vcover merge](#), [vcover ranktest](#)

coverage save

The **coverage save** command is used to save the coverage results of the specified type from a simulation to the unified coverage database (UCDB). If no type is specified, then all types are saved into the database.

While code coverage data can also be saved with the **\$coverage_save** system task (see [System Tasks and Functions](#) in the User's Manual), the coverage save command is the preferred method of saving coverage data.

The report displays code coverage data from generate blocks.

Syntax

```
coverage save [-instance <path>] [-code {b | c | e | f | s | t}...] [-codeAll]
              [-du <du_name>] [-testname <test_name>] [-norecursive] [-onexit] <file_name>]
              [-assert] [-coveredfcov] [-cvg] [-directive]
```

Arguments

- -assert
(optional) Save only assertion coverage data.
- -code {b | c | e | f | s | t}...
(optional) Save only the designated coverage type: b=branch coverage; c=condition coverage; e=expression coverage; f=Finite State Machine coverage; s=statement coverage; t=toggle. More than one coverage type can be specified with a single -code argument (example: “-code bces”).
- -codeAll
(optional) Specifies the command apply to all coverage types. Equivalent to -code bcestf or no specification of type at all.
- -coveredfcov
(optional) Save only non-zero covergroup bins and covered cover directives into the UCDB. Only supported during live simulations, not in Coverage View mode.
- -cvg
(optional) Save only covergroup coverage data.
- -directive
(optional) Save only cover directives coverage data.
- -du <du_name>
(optional) Saves coverage statistics for the specified design unit. Only supported during live simulation, not in Coverage View mode.

<du_name> is <library name>.<primary>(<secondary>), where the library name is optional, and secondary name is required only for VHDL. If there are parameterized instances, all are considered to match the specified design unit.

- `-instance <path>`
(optional) Saves coverage data for only a specified instance and any of its children, recursively. Use the `-norecursive` argument to exclude data from instance children. `<path>` is a path to the instance. You can specify more than one instance during live simulation but only one instance can be specified in Coverage View mode. `<path>` can also be used to specify a dataset other than the current dataset. (See Object Name Syntax for instructions on how to specify a dataset.) If no dataset is specified, the current dataset is used. Only one dataset name per command invocation may be used or an error will result.
- `-norecursive`
(optional) Saves coverage data while excluding data from children of the specified instance or design unit.
- `-onexit`
(optional) Causes ModelSim to save coverage data automatically when the simulator exits.
- `-testname <test_name>`
(optional) Saves coverage data with the specified test name, instead of the base name of the UCDB file being saved. In Coverage View mode, this argument can only be used with a simulation-generated UCDB or a UCDB file containing only a single test record.
- `<file_name>`
(required) Designates the name of the database file to save. Required.

Examples

- Save data from the current simulation into *myfile1.ucdb*:

```
coverage save myfile1
```
- Save data from current simulation (into *somefile.ucdb*) when the simulator exits:

```
coverage save -onexit somefile
```
- Save data for a specific design unit or instance in the design and all its children:

```
coverage save -instance ./path/inst1 mycov
```
- Save UCDB (*mycov*) which contains *./path/inst1* hierarchy with a testname of *mytest*:

```
coverage save -instance ./path/inst1 -testname mytest mycov
```

See also

[Code Coverage](#), [“Coverage and Verification Management in the UCDB”](#), [coverage attribute](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage analyze](#), [vcover merge](#), [vcover ranktest](#)

coverage tag

The **coverage tag** command is used to create a "link" between different objects in the coverage database, or to link specific coverage objects to records in a testplan. A tag is a simple string associated with the object. Objects sharing the same tag name are linked together. Once an object is tagged, its children will not be tagged.

This command is only available during post-simulation processing, when a UCDB file is opened with **vsim -viewcov**.

You can add multiple tags on any object to mark it as a member of different categories.

Tags can be used for primitive (non-hierarchical) grouping of coverage in the database.

If you issue the coverage tag command without the **-tagname** switch, it returns a list of all tags, which is essentially a general purpose hierarchy browser. See examples below.

The **-cvgmatch**, **-cvginstmatch**, **-cvpmatch** switches are implicitly recursive and can not be used with coverage type filters (-assert, -codeAll, -cvg, etc.). If the **-cvgmatch**, **-cvginstmatch**, **-cvpmatch** switches are used together, the lowest-level object is tagged, but the higher-level matches must be satisfied, too, before they are tagged. Thus, the matching requirements of the multiple arguments are effectively ANDed. The order of matching from lowest to highest are: -cvpmatch, -cvginstmatch, -cvgmatch. See examples below.

Syntax

For Linking/Unlinking Any Type of Coverage

```
coverage tag  
  [-assert] [-code {b | c | e | f | s | t}...] [-codeAll] [-cvg]  
  [-directive] [-duname <duname>]  
  [[-match <string>] | [-recursive]]  
  [-path <path> +] [-plansection <path>] [-select instance]  
  [-tagname <name> [-delete]]
```

For Linking/Unlinking covergroups, covergroup instances, coverpoints, and crosses

```
coverage tag  
  [-cvgmatch <string>] [-cvginstmatch <string>] [-cvpmatch <string>]  
  [-duname <duname>]  
  [-path <path> +] [-plansection <path>]  
  [-tagname <name> [-delete]]
```

For Linking/Unlinking Directed Tests

```
coverage tag  
  [-tagname <name> [-delete]] [-testrecord <test_name> [-delete]]
```

Arguments

- -assert
(optional) Specifies the command apply to assertion data.

- `-code {b | c | e | f | s | t}...`
(optional) Specifies this command apply to corresponding code coverage types: branch, condition, expression, statement, toggle, FSM. More than one coverage type can be specified with a single `-code` argument (example: “`-code bces`”).
- `-codeAll`
(optional) Specifies the command apply to all coverage types. Equivalent to `-code bcestf`.
- `-cvg`
(optional) Specifies the command apply to covergroup data.
- `-cvgmatch <string>`
(optional) Specifies a match of a covergroup type against the specified pattern. This argument is implicitly recursive. If used with `-cvginstmatch` and/or `-cvpmatch`, the lowest-level object is tagged, but the higher-level matches must be satisfied, too, for the tagging to be successful.
- `-cvginstmatch <string>`
(optional) Specifies a match of a covergroup instance name (`<option>.<name> <value>`) against the specified instance. Wildcards are allowed.
- `-cvpmatch <string>`
(optional) Specifies a match of a coverpoint or cross name against the specified covergroup instance or covergroup type. Wildcards are allowed.
- `-delete`
(optional) Removes the given tags from the objects specified by the **-tagname** or **-testname** switch.
- `-directive`
(optional) Specifies the command apply to directive data.
- `-duname <duname>`
(optional) Specifies the command apply to the specified design unit(s). Wildcards are allowed.
- `-match <string>`
(optional) Recursively matches the given pattern against the specified coverage types in the entire instance tree. If `-duname` is specified, it matches against the specified coverage types in the design unit. Wildcards are allowed. This switch is mutually exclusive with **-recursive**.
- `-path <path> +`
(optional) Specifies a path to which the tag is applied in the UCDB. More than one path may be specified. Wildcards are allowed. You may specify “-” to indicate that the path should be ignored. Useful when path is specified in the Link column of a verification plan. For more information, see the “[xml2ucdb.ini Configuration File](#)” appendix. `<path>` may also be used

to specify a dataset other than the current dataset. (See Object Name Syntax for instructions on how to specify a dataset.) If no dataset is specified, the current dataset is used. Only one dataset name per command invocation may be used or an error will result.

- `-plansection <path>`
(optional) Specifies a test plan path to which the tag is applied. Wildcards are allowed.
- `-recursive`
(optional) Recursively selects objects under the given scope. Requires one of the [coverage types] options. The following command:

```
-recursive -path p -cvg
```

behaves as if `-path p/*`, `-path p/*/*`, `-path p/*/*/*`, etc. were given for as many levels as are applicable. This switch is mutually exclusive with **-match**.



Tip: When writing a tag, once a scope is tagged, none of its children or descendants will be; this is because coverage numbers always include descendants, so tagging descendants adds redundant coverage to a testplan.

- `-select instance`
(optional) Specifies the command applies to instance scopes HDL instance scopes (VHDL architectures, interface instances, etc.).
- `-tagname <name>`
(optional) Specifies the name of the tag. May be specified multiple times or not at all. When not specified, a list of all tags is printed.
- `-testrecord <test_name>`
(optional) Specifies the name of the specific test record in the UCDB to be linked. This tag creates a link between an item in the test plan and the test record (name of individual test run) in the UCDB. Wildcards are allowed. Can only be used in conjunction with **-tagname** and **-delete**; mutually exclusive with all other arguments.

Examples

- Display all covergroup scopes, recursively, in the entire design:

```
coverage tag -path / -r -cvg
```

Returns the following:

```
# /coverpkg/statecover [covergroup]: "2.2"  
# /coverpkg/statecover/cvpi [coverpoint] (no tags)  
# /coverpkg/statecover/cvpstate [coverpoint] (no tags)  
# /coverpkg/statecover/i_x_state [cross] (no tags)
```

- Tag a given covergroup `"/a/b/cvg0"` and a given instance `"/a/d"`:

```
coverage tag -tagname T1 -path /a/b/cvg0 /a/d
```

- Tag all covergroups in the instance /a/b:
`coverage tag -tagname T1 -cvg -path /a/b/*`
- Tag all covergroups in the subtree rooted at /a/b:
`coverage tag -tagname T1 -recursive -cvg -path /a/b`
- Tag all toggles whose names start with "d" inside the given design unit:
`coverage tag -tagname T1 -code t -duname duname -path d*`
- Tag all objects matching the pattern in all design units
`coverage tag -tagname T1 -match a* -du *`
- Tag all branches in the given design unit; this implicitly recurses all paths in the design unit because the filter is given and no path with the -duname:
`coverage tag -tagname T1 -code b -duname work.duname`
- Tag all branches in any test data records that start with the pattern "sample":
`coverage tag -tagname T1 -testrecord sample*`
- Tag all covergroups whose type names start with the pattern "xactor":
`coverage tag -tagname T1 -cvgmatch xactor*`
- Almost the same, but will tag coverpoint, cross, and covergroup instance scopes matching the pattern:
`coverage tag -tagname T1 -cvg -match xactor*`
- Tag a coverpoint named "cvp1" that belongs to coverage instance "cvginst1" in covergroup "cvgl"; note this will not tag coverpoints named "cvp1" that happen to exist in other contexts. Wildcard patterns are accepted, too, in which case any coverpoint-coverinstance-covergroup combination that matches are tagged as well:
`coverage tag -tagname T1 -cvgmatch cvgl -cvginstmatch cvginst1
-cvpmatch cvp1`
- Tag a coverpoint or cross matching the given pattern inside a covergroup type named "cvgtype":
`coverage tag -tagname T1 -cvgmatch cvgtype -cvpmatch *myvar*`

See also

[“Code Coverage”](#), [“Verification Browser Window”](#), [“Verification Tracker Window”](#), [“xml2ucdb.ini Configuration File” appendix](#), [coverage analyze](#), [coverage attribute](#), [coverage exclude](#), [coverage ranktest](#), [coverage goal](#), [coverage report](#), [coverage save](#), [coverage unlinked](#), [vcover merge](#), [vcover ranktest](#)

coverage testnames

The **coverage testnames** command displays the testnames in the UCDB file currently loaded into memory. If a merged file, it gives you a list of tests in the merged file.

This command is most useful if you use the `-testextract` of `coverage analyze` or `coverage report`, because it requires the test name. By default, the testname is the name of the UCDB file, though you can set it to whatever you would like. Set the test name, before saving the UCDB file, using the command `"coverage attribute -test mytestname"`.

This command is only available during post-simulation processing, when a UCDB file is opened with **`vsim -viewcov`**.

Syntax

`coverage testnames [-tcl]`

Arguments

- `-tcl`
(optional) Print attribute information in a tcl format.

See also

[Code Coverage](#), [“Verification Browser Window”](#), [coverage attribute](#), [coverage exclude](#), [coverage goal](#), [coverage ranktest](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage analyze](#), [coverage weight](#), [vcover merge](#), [vcover ranktest](#), [vcover stats](#), [vcover testnames](#)

coverage unlinked

The **coverage unlinked** command is used as a diagnostic tool to report on:

- test plan item with no coverage associated
- directed tests within a testplan with no coverage associated
- design coverage that has not been associated with an item in a test plan

It reports on objects that are "unlinked" for the purposes of test traceability.

This command is only available during post-simulation processing, when a UCDB file is opened with **vsim -viewcov**.

i **Tip:** When applying this command to design paths, it is a good idea to restrict the command to types of objects that must be linked (only covergroups and cover directives, for example). Otherwise, the command can report as "unlinked" many irrelevant items.

Syntax

```
coverage unlinked
  [-recursive] [-plansection <path>] [-testname <test_name>] [-du <duname>] [-path <path>]
  [<coverage_type>] [-select instance] [-file <filename>] [-append]

<coverage type> =
  [-assert] [-code {b | c | e | f | s | t}...] [-codeAll] [-cvg] [-directive]
```

Arguments

- **-append**
(optional) Appends to the output file, if **-file** option is used.
- **-assert**
(optional) Specifies the command apply to assertion data.
- **-code {b | c | e | f | s | t}...**
(optional) Specifies this command apply to corresponding code coverage types: branch, condition, expression, statement, toggle, FSM. More than one code coverage type can be specified with each **-code** argument (example: “-code bces”).
- **-codeAll**
(optional) Specifies the command apply to all coverage types. Equivalent to **-code bcestf**.
- **-cvg**
(optional) Specifies the command apply to covergroup data.
- **-directive**
(optional) Specifies the command apply to directive data.

- `-du <duname>`
(optional) Restricts the report to design units matching the given specification, `<duname>`. Wildcards are allowed.
- `-file <filename>`
(optional) Write to output file.
- `-path <path>`
(optional) Restricts unlinked results to design (non-testplan) paths matching the specified path. This argument can be combined with `-du` to be du-relative. `<path>` may also be used to specify a dataset other than the current dataset. (See Object Name Syntax for instructions on how to specify a dataset.) If no dataset is specified, the current dataset is used. Only one dataset name per command invocation may be used or an error will result.
- `-plansection <path>`
(optional) Restricts the command to the given testplan path. By default, the query treats all testplan scopes in the database, as well as all linked scopes associate with the testplan scopes.
- `-recursive`
(optional) Specifies that the command is applied recursively. The default is for the query to be restricted to the single object or objects specified in the command.
- `-select instance`
(optional) Restricts the command to apply to design instance scopes (VHDL architectures, interface instances, etc.).
- `-testname <test_name>`
(optional) Finds unlinked tests within a testplan. Wildcards are allowed. Use this argument to ensure that a directed test ran within a testplan.

Examples

- Find testplan sections which are unlinked — testplans that do not share a tag with a non-testplan scope or vice-versa:

```
coverage unlinked -plansection / -r
```
- Find any unlinked covergroup or cover directive:

```
coverage unlinked -path / -cvg -dir -r #
```
- Find design units that are unlinked:

```
coverage unlinked -du *
```
- Find everything under an instance tree that is unlinked, does not include design units:

```
coverage unlinked -path / -r
```


See also

[Code Coverage](#), [“Verification Browser Window”](#), [coverage analyze](#), [coverage attribute](#), [coverage exclude](#), [coverage goal](#), [coverage ranktest](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage testnames](#), [coverage weight](#), [vcover merge](#), [vcover ranktest](#), [vcover stats](#)

coverage weight

The **coverage weight** command sets a global per-type weight for total coverage calculations.

Total coverage calculations are exposed with the [coverage analyze -total](#) command; this includes total coverage numbers for test plans in the Test Tracking tab in the “[Verification Browser Window](#)” or “[Verification Tracker Window](#)”,

Specifically, the command sets the overall weight for covergroups (by instance, or by design unit) and weights for individual items (design units, instances, and/or cover directives, etc.). Use the `-plansection`, `-path`, and `-du` arguments to set the weights for individual coverage items, design instances, design units, covergroups, or test plan items. Setting weights for individual items affects coverage the same as `option.weight` or `type_option.weight`.

For more information regarding the use of the arguments and the type coverage statistics gathered, see [Calculation of Total Coverage](#).

Syntax

Setting global per-type weights

```
coverage weight [-bydu] [-byinstance] [-type] [-fstate] [-ftrans]
  [-fail] [-pass] [-vpass] [-disabled] [-attempted] [-active]
  [-cvg] [-code {b | c | e | f | s | t}...] [-codeAll] [-directive]
  [<integer_weight>]
```

Setting weight for individual objects — used when objects are part of a verification (test) plan

```
coverage weight {-du <du_name> | -path <path> | -plansection <section_name>}
  <integer_weight>
```

Arguments

- Assertion directive arguments are as follows.
 - active
Active assertion directives, per instance. Optional.
 - fail
Assertion directive failures, per instance. Optional.
 - pass
Assertion directive passes, per instance. Optional.
 - vpass
Assertion directive vacuous passes, per instance. Optional.

Note



These have no effect in the simulator because these do not contribute to coverage calculations. They are still stored in the database, however.

- -bydu
(optional) Modifier used to set per-du (code coverage only)
- -byinstance
(optional) Modifier used to set a per-instance goal (code coverage and covergroup).
- -code {b | c | e | f | s | t}...
(optional) Sets weight for code coverage data for coverage type: b=branch coverage; c=condition coverage; e=expression coverage; s=statement coverage; t=toggle; f=Finite State Machine coverage. More than one coverage type can be specified in a single -code argument (example: “-code bces”).
- -codeAll
(optional) Specifies the command for all coverage types. Equivalent to -code bcestf.
- -cvg
(optional) Sets global weight for covergroup data only.
- -directive
(optional) Sets global weight for directive data only.
- -du <du_name>
(optional) Sets the weight for a given design unit. Mutually exclusive with **-path** and **-plansection**. Cannot be combined with any other arguments besides **<integer_weight>**.
- -fstate
(optional) Selects FSM state coverage.
- -ftrans
(optional) Selects FSM transition coverage.
- <integer_weight>
(required in order to set weights; if left unspecified, command prints weights) Specifies the value for the weight: must be a natural integer, greater than or equal to 0. A weight of 0 turns off the coverage summary for the specified item or covergroup.
- -path <path>
(optional) Sets the weight for a given coverage/design object. Mutually exclusive with **-du** and **-plansection**. Cannot be combined with any other arguments besides **<integer_weight>**. <path> may also be used to specify a dataset other than the current dataset. If no dataset is specified, the current dataset is used. Only one dataset name per command invocation may be used or an error will result.
- -plansection <section_name>
(optional) Sets the weight for a given test plan section. Mutually exclusive with **-du** and **-path**. Cannot be combined with any other arguments besides **<integer_weight>**.

- -type
(optional) Specifies the command for covergroup type coverage.

See also

[Code Coverage, Coverage and Verification Management in the UCDB](#)", [“Verification Browser Window”](#), [“Verification Tracker Window”](#), [coverage analyze](#), [coverage attribute](#), [coverage exclude](#), [coverage goal](#), [coverage ranktest](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage testnames](#), [coverage unlinked](#)

dataset alias

This command maps an alternate name (alias) to an open dataset. A dataset can have any number of aliases, but all dataset names and aliases must be unique even when more than one dataset is open. Aliases are not saved to the *.wlf* file and must be remapped if the dataset is closed and then re-opened.

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

```
dataset alias <dataset_name> [<alias_name>]
```

Arguments

- <dataset_name>
(required) Specifies a dataset name or currently assigned dataset alias. Must be specified as the first argument to the **dataset alias** command. Returns a list of all aliases mapped to the specified dataset file when specified without <dataset_alias>.
- <alias_name>
(optional) Specifies string to assign to the dataset as an alias. Wildcard characters are permitted.

Examples

Assign the alias name “bar” to the dataset named “gold.”

```
dataset alias gold bar
```

Related Topics

- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)

dataset clear

This command applies only to WLF based simulation datasets. It has no effect on coverage (UCDB) datasets. All event data is removed from the current simulation WLF file, while retaining all currently logged signals. Subsequent run commands will continue to accumulate data in the WLF file.

If the command is executed when no design is loaded then the error: “Dataset not found:sim” is returned. If the command is executed when a design is loaded, then the “sim:” dataset is cleared, irrespective of which dataset is currently set. Clearing the dataset will clear any open wave window based on the “sim:”.

Syntax

dataset clear

Examples

Clear data in the WLF file from time 0ns to 100000ns, then log data into the WLF file from time 100000ns to 200000ns.

```
add wave *  
run 100000ns  
dataset clear  
run 100000ns
```

Related Topics

- [dataset alias](#)
- [dataset close](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)
- [log](#)
- [Recording Simulation Results With Datasets](#)

dataset close

This command closes an active dataset. To open a dataset, use the [dataset open](#) command.

Syntax

```
dataset close {<dataset_name> | -all}
```

Arguments

- <dataset_name> | -all
(required) Closes active dataset(s).
 <dataset_name> — Specifies the name of the dataset or alias you wish to close.
 -all — Closes all open datasets and the simulation.

Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)

dataset config

This command configures WLF parameters for an open dataset and all aliases mapped to that dataset. It has no effect on coverage datasets (UCDB).

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

```
dataset config <dataset_name> [-wlf cachesize [<n>]] [-wlf deleteonquit [0 | 1]] [-wlf opt [0 | 1]]
```

Arguments

- <dataset_name>
(required) Specifies a open dataset or dataset alias you wish to configure. Must be specified as the first argument to the **dataset config** command.
- -wlf cachesize [<n>]
(optional) Sets the size, in megabytes, of the WLF reader cache. Does not affect the WLF write cache.

 <n> — Any non-negative integer, in MB where the default is 256.

If you do not specify a value for <n>, this switch returns the size, in megabytes, of the WLF reader cache.
- -wlf deleteonquit [0 | 1]
(optional) Deletes the WLF file automatically when the simulation exits. Valid for the current simulation dataset only.

 0 — Disabled (default)
 1 — Enabled

If you do not specify an argument, this switch returns the current setting for the switch.
- -wlf opt [0 | 1]
(optional) Optimizes the display of waveforms in the Wave window.

 0 — Disabled
 1 — Enabled (default)

If you do not specify an argument, this switch returns the current setting for the switch.

Examples

Set the size of the WLF reader cache for the dataset “gold” to 512 MB.

```
dataset config gold -wlf cachesize 512
```


Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)
- [WLF File Parameter Overview](#)
- [vsim](#)

dataset current

This command activates the specified dataset and sets the GUI context to the last selected context of the specified dataset. All context dependent GUI data is updated and all context dependent CLI commands start working with respect to the new context.

Syntax

```
dataset current [<dataset_name>]
```

Arguments

- <dataset_name>
(optional) Specifies the dataset name or dataset alias you want to activate. If no dataset name or alias is specified, the command returns the name of the currently active dataset.

Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)
- [WLF File Parameter Overview](#)
- [vsim](#)

dataset info

This command reports a variety of information about a dataset.

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

```
dataset info {name | file | exists} <dataset_name>
```

Arguments

- {name | file | exists}

(required) Identifies what type of information you want reported.

Only one option per command is allowed. The current options include:

name — Returns the name of the dataset. Useful for identifying the real dataset name of an alias.

file — Returns the name of the WLF file or UCDB file associated with the dataset.

exists — Returns "1" if the dataset is currently open, "0" if it does not.

Must be specified as the first argument to the **dataset info** command.

- <dataset_name>

(optional) Specifies the name of the dataset or alias for which you want information. If you do not specify a dataset name, ModelSim uses the dataset of the current environment.

Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)
- [environment](#)

dataset list

This command lists all active datasets.

Syntax

```
dataset list [-long]
```

Arguments

- `-long`
(optional) Lists the dataset name followed by the *.wlf* file to which the dataset name is mapped.

Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset info](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)

dataset open

This command opens a WLF file (representing a prior simulation) and/or UCDB file (representing coverage data) and assigns it the logical name that you specify. To close a dataset, use [dataset close](#).

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

```
dataset open <file_name> [<dataset_name>]
```

Arguments

- <file_name>
(required) Specifies the WLF file or UCDB file to open as a view-mode dataset. Must be specified as the first argument to the **dataset open** command.
- <dataset_name>
(optional) Specifies a name for the open dataset. This is a name that will identify the dataset in the current session. By default the dataset prefix will be the name of the specified WLF or UCDB file.

Examples

Open the dataset file *last.wlf* and assign it the name *test*.

```
dataset open last.wlf test
```

Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)
- [vsim -view option](#)

dataset rename

This command changes the name of a dataset to the new name you specify.

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

```
dataset rename <dataset_name> <new_dataset_name>
```

Arguments

- <dataset_name>
Specifies the existing name of the dataset.
- <new_dataset_name>
Specifies the new name for the dataset.

Examples

Rename the dataset file "test" to "test2".

```
dataset rename test test2
```

Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)

dataset restart

This command unloads the specified dataset or currently active dataset and reloads the dataset using the same dataset name. The contents of Wave and other coverage windows are restored for UCDB datasets after a reload.

Syntax

```
dataset restart [<file_name>]
```

Arguments

- <file_name>

(optional) Specifies the WLF or UCDB file to open as a view-mode or coverage mode dataset. If <filename> is not specified, the currently active dataset is restarted.

Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset save](#)
- [dataset snapshot](#)

dataset save

This command writes data from the current simulation to the specified file. This lets you save simulation data while the simulation is still in progress.

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

This command is equivalent to the [coverage save](#) command for coverage datasets.

Syntax

```
dataset save <dataset_name> <file_name>
```

Arguments

- <dataset_name>
(required) Specifies the name of the dataset you want to save.
- <file_name>
(required) Specifies the name of the file to save.

Examples

Save all current log data in the sim dataset to the file *gold.wlf*.

```
dataset save sim gold.wlf
```

Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset snapshot](#)

dataset snapshot

This command saves data from the current WLF file (*vsim.wlf* by default) at a specified interval. It provides you with sequential or cumulative "snapshots" of your simulation data. This command does not apply to coverage datasets (UCDB).

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

```
dataset snapshot [-dir <directory>] [-disable] [-enable] [-file <file_name>]
  [-filemode {overwrite | increment}] [-mode {cumulative | sequential}] [-report] [-reset]
  {-size <file_size> | -time <n> [<unit>]}
```

Arguments

- **-dir <directory>**
(optional) Specifies a directory into which the files should be saved. Either absolute or relative paths may be used. Default is to save to the current working directory.
 - **-disable**
(optional) Turns snapshotting off. All dataset snapshot settings from the current simulation are stored in memory. All other options are ignored after you specify **-disable**.
 - **-enable**
(optional) Turns snapshotting on. Restores dataset snapshot settings from memory or from a saved dataset. (default)
 - **-file <file_name>**
(optional) Specifies the name of the file to save snapshot data.

 <file_name> — A specified file name where the default is *vsim_snapshot.wlf*. The suffix *.wlf* will be appended to specified filename and, possibly, an incrementing suffix.
- When the duration of the simulation run is not a multiple of the interval specified by **-size** or **-time**, the incomplete portion is saved in the file *vsim.wlf*.
- **-filemode {overwrite | increment}**
(optional) Specifies whether to overwrite the snapshot file each time a snapshot occurs.

 overwrite — (default)

 increment — A new file is created for each snapshot. An incrementing suffix (1 to n) is added to each new file (for example, *vsim_snapshot_1.wlf*).
 - **-mode {cumulative | sequential}**
(optional) Specifies whether to keep all data from the time signals are first logged.

 cumulative — (default)

 sequential — The current WLF file is cleared every time a snapshot is taken.

dataset snapshot

- **-report**
(optional) Lists current snapshot settings in the Transcript window. All other options are ignored if you specify **-report**.
- **-reset**
(optional) Resets values back to defaults. The behavior is to reset to the default, then apply the remainder of the arguments on the command line. See examples below. If specified by itself without any other arguments, **-reset** disables dataset snapshot and resets the values.
- **-size <file_size>**
(Required if **-time** is not specified.) Specifies that a snapshot occurs based on WLF file size. Must be specified as the final argument to the **dataset snapshot** command.
<file_size> — Size of WLF file in MB.
- **-time <n> [<unit>]**
(Required if **-size** is not specified.) Specifies that a snapshot occurs based on simulation time. Must be specified as the final argument to the **dataset snapshot** command.
<n> — Any positive integer.
<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<limit>** and **<unit>** within curly braces ({}).

Examples

- Create the file *vsim_snapshot_<n>.wlf* that is written to every time the current WLF file reaches a multiple of 10 MB (i.e., at 10 MB, 20 MB, 30 MB, etc.).

dataset snapshot -size 10

- Similar to the previous example, but in this case the current WLF file is cleared every time it reaches 10 MB.

dataset snapshot -size 10 -mode sequential

- Assuming simulator time units are ps, this command saves a file called *gold_<n>.wlf* every 1000000 ps. If you run the simulation for 3000000 ps, three files are saved: *gold_1.wlf* with data from 0 to 1000000 ps, *gold_2.wlf* with data from 1000001 to 2000000, and *gold_3.wlf* with data from 2000001 to 3000000.

**dataset snapshot -time 1000000 -file gold.wlf -mode sequential
-filemode increment**

Because this example sets the time interval to 1000000 ps, if you run the simulation for 3500000 ps, a file containing the data from 3000001 to 3500000 ps is saved as *vsim.wlf* (default).

- Enable snapshotting with time=10000 and default mode (cumulative) and default filemode (overwrite).

dataset snapshot -reset -time 10000

Related Topics

- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)

delete

This command removes objects from either the List or Wave window. Arguments to this command are order dependent.

Syntax

```
delete list [-window <wname>] <object_name>
```

```
delete wave [-window <wname>] <object_name>
```

Arguments

- list
Specifies the target is a list window.
- wave
Specifies the target is a wave window.
- -window <wname>
(optional) Specifies the name of the List or Wave window to target for the delete command. (The [view](#) command allows you to create more than one List or Wave window.) If no window is specified, the default window is used; the default window is determined by the most recent invocation of the view command.
- <object_name>...
(required) Specifies the name of an object. Must match an object name used in an [add list](#) or [add wave](#) command. Multiple object names are specified as a space separated list. Wildcard characters are allowed. Must be specified as the final argument to the **delete list** and **delete wave** commands.

Examples

- Remove the object *vec2* from the list2 window.

```
delete list -window list2 vec2
```
- Remove all objects beginning with the string */test* from the Wave window.

```
delete wave /test*
```

Related Topics

- [add list](#)
- [add wave](#)
- [Wildcard Characters](#)

describe

This command displays information about the following types of simulation objects and design regions in the Transcript window:

- **VHDL** — signals, variables, and constants
- **Verilog** — nets and registers
- **C** — variables
- **SystemC** — signals, ports, FIFOs, and member variables of modules
- Design region

VHDL signals, Verilog nets and registers, and SystemC signals and ports can be specified as hierarchical names.

C variables can be described if you are running “[C Debug](#)”, and the variables are local to the active call frame for the line in the function in the C source file where you are stopped.

For specific information related to viewing SystemC objects refer to “[SystemC Object and Type Display](#)”.

Syntax

```
describe <name>...
```

Arguments

- <name>...
(required) The name of an HDL object, SystemC signal, or C variable for which you want a description. Multiple object names are specified as a space separated list. Wildcard characters are allowed. HDL object names can be relative or full hierarchical names.

Examples

- Print the type of C variable *x*.
describe x
- Print the type of what *p* points to.
describe *p
- Print the types of the three specified signals.
describe clk prw prdy

Related Topics

- [add list](#)
- [add wave](#)
- [Wildcard Characters](#)

disablebp

This command turns off breakpoints and [when](#) commands. To turn on breakpoints or when commands again, use the [enablebp](#) command.

Syntax

```
disablebp [<id#> | <label>]
```

Arguments

- [<id#>](#)
(optional) Specifies the ID number of a breakpoint or when statement to disable.
Note that C breakpoint id#s are prefixed with "c."
- [<label>](#)
(optional) Specifies the label name of a breakpoint or when statement to disable.

If you do not specify either of these arguments, all breakpoints and when statements are disabled.

Use the `bp` command with no arguments to find labels and ID numbers for all breakpoints in the current simulation. Use the `when` command with no arguments to find labels and ID numbers of all when statements in the current simulation.

Note



Id numbers for breakpoints and when statements are assigned from the same pool. Even if you have not specified a given id number for a breakpoint, that number may still be used for a when command.

Related Topics

- [bd](#)
- [bp](#)
- [C Debug](#)
- [enablebp](#)
- [onbreak](#)
- [resume](#)
- [when](#)

disable_menu

This command disables the specified menu within the specified window.

The disabled menu will become grayed-out and nonresponsive.

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

```
disable_menu <window_name> <menu_path>
```

Arguments

- <window_name>
(required) The path of the window containing the menu. You must express the path for the Main window as "". All other window pathnames begin with a period (.) as shown in the examples below.
- <menu_path>
(required) Name of the Tk menu-widget path. Must be specified as the final argument to the **disable_menu** command.

Examples

- Disable the file menu of the Main window.

```
disable_menu "" File
```
- Disable the file menu of the mywindow window.

```
disable_menu .mywindow File
```

Related Topics

- [add_menu](#)
- [disable_menuitem](#)
- [enable_menu](#)

disable_menuitem

This command disables a specified menu item within the specified menu path of the specified window.

The menu item will become grayed-out and nonresponsive.

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

```
disable_menuitem <window_name> <menu_path> <label>
```

Arguments

- <window_name>
(required) Tk path of the window containing the menu.
Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.
- <menu_path>
(required) Name of the Tk menu-widget path. The path may include a submenu as shown in the example below.
- <label>
(required) Menu item text.

Examples

- Locate the mywindow window, disable the Save Results As... menu item in the save submenu of the file menu.

```
disable_menuitem .mywindow file.save "Save Results As..."
```

Related Topics

- [add_menu](#)
- [disable_menu](#)
- [enable_menu](#)

do

This command executes the commands contained in a macro file.

A macro file can have any name and extension. An error encountered during the execution of a macro file causes its execution to be interrupted, unless an [onerror](#) command, [onbreak](#) command, or the `OnErrorDefaultAction` Tcl variable has specified with the [resume](#) command.

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

```
do <filename> [<parameter_value>...]
```

Arguments

- <filename>
(required) Specifies the name of the macro file to be executed. The name can be a pathname or a relative file name. Pathnames are relative to the current working directory. Must be specified as the first argument to the **do** command.

If the `do` command is executed from another macro file, pathnames are relative to the directory of the calling macro file. This allows groups of macro files to be stored in a separate sub-directory.

- <parameter_value>...
(optional) Specifies values that are to be passed to the corresponding parameters \$1 through \$9 in the macro file. Multiple parameter values must be separated by spaces.

If you want to make the parameters optional (for example, specify fewer parameter values than the number of parameters actually used in the macro), you must use the [argc](#) simulator state variable in the macro. Refer to “[Making Macro Parameters Optional](#)”.

Note



While there is no limit on the number of parameters that can be passed to macros, only nine values are visible at one time. Use the [shift](#) command to see the other parameters.

Examples

- Execute the file *macros/stimulus* and pass the parameter value 100 to \$1 in the macro file.

```
do macros/stimulus 100
```

- Where the macro file *testfile* contains the line

```
bp $1 $2
```

place a breakpoint in the source file named *design.vhd* at line 127.

do testfile design.vhd 127

Related Topics

- [Tcl and Macros \(DO Files\)](#)
- [Modes of Operation](#)
- [Using a Startup File](#)
- [DOPATH variable](#)
- [Saving a Transcript File as a Macro \(DO file\)](#)

down

This command searches for object transitions or values in the specified List window.

It executes the search on objects currently selected in the window, starting from the point of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which an object takes on a particular value, or an expression of multiple objects evaluates to true. See the [up](#) command for related functionality.

The procedure for using down includes three steps:

1. Click on the desired object.
2. Click on the desired starting location.
3. Issue the down command. (The [seetime](#) command can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

Syntax

```
down [-expr <expression>] [-falling] [<n>] [-noglitch] [-rising] [-value <sig_value>]  
    [-window]
```

Arguments

- -expr <expression>
(optional) The List window is searched until the expression evaluates to a boolean true condition.

 <expression> — An expression that involves one or more objects, but limited to objects that have been logged in the referenced List window. An object may be specified either by its full path or by the shortcut label displayed in the List window.

See [GUI_expression_format](#) for the format of the expression. The expression must be placed within curly braces.
- -falling
(optional) Searches for a falling edge on the specified object if that object is a scalar object. If it is not a scalar object, the option will be ignored.
- <n>
(optional) Specifies to find the nth match. If less than n are found, the number found is returned with a warning message, and the marker is positioned at the last match.
- -noglitch
(optional) Specifies that delta-width glitches are to be ignored.

down

- **-rising**
(optional) Searches for a rising edge on the specified object if that object is a scalar object. If it is not a scalar object, the option will be ignored.
- **-value <sig_value>**
(optional) Specifies the value of the object to match.
<sig_value> — A value specified in the same radix that the selected object is displayed. Case is ignored, but otherwise the value must be an exact string match. We do not support don't-care bits.
- **-window**
(optional) Specifies an instance of the List window that is not the default. When **<wname>** is not specified, the default List window is used. Use the [view](#) command to change the default window.
<wname> — The name of a List window not currently the default.

Examples

- Find the next time at which the selected vector transitions to FF23, ignoring glitches.
down -noglitch -value FF23
- Go to the next transition on the selected object.

down

The following examples illustrate search expressions that use a variety of object attributes, paths, array constants, and time variables. Such expressions follow the [GUI_expression_format](#).

- Search down for an expression that evaluates to a boolean 1 when object *clk* just changed from low to high and object *mystate* is enumeration reading and object */top/u3/addr* is equal to the specified 32-bit hex constant.
down -expr {clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)}
- Search down for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit object */top/u3/addr* equals hex ac.
down -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
- Search down for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, clock just changed from low to high, and object *mode* is enumeration writing.
down -expr {(NOW > 23 us) && (NOW < 54 us) && clk'rising && (mode == writing)}

Related Topics

- [GUI_expression_format](#)
- [view](#)
- [seetime](#)
- [up](#)

drivers

This command displays the names and strength of all drivers of the specified object.

The driver list is expressed relative to the top-most design signal/net connected to the specified object. If the object is a record or array, each sub-element is displayed individually.

Syntax

```
drivers <object_name>
```

Arguments

- <object_name>
(required) Specifies the name of the signal or net whose drivers are to be shown. All signal or net types are valid. Multiple names and wildcards are accepted.

Example

```
drivers /top/dut/pkt_cnt(4)
```

```
# Drivers for /top/dut/pkt_cnt(4):  
#   St0 : Net /top/dut/pkt_cnt[4]  
#       St0 : Driver /top/dut/pkt_counter/#IMPLICIT-WIRE(cnt_out)#6
```

In some cases, the output may supply a strength value similar to 630 or 52x, which indicates an ambiguous verilog strength.

Related Topics

- [readers](#)
- Verilog LRM Std 1365-2005 section 7.10.2 "Ambiguous strengths: sources and combinations"

dumplog64

This command dumps the contents of the specified WLF file in a readable format to stdout.

The WLF file cannot be opened for writing in a simulation when you use this command. This command cannot be used in a DO file.

Syntax

```
dumplog64 <filename>
```

Arguments

- <filename>
(required) The name of the WLF file to be read.

echo

This command displays a specified message in the Transcript window.

Syntax

```
echo [<text_string>]
```

Arguments

- <text_string>
(required) Specifies the message text to be displayed. If the text string is surrounded by quotes, blank spaces are displayed as entered. If quotes are omitted, two or more adjacent blank spaces are compressed into one space.

Examples

- If the current time is 1000 ns, this command:

```
echo "The time is    $now ns."
```

returns the message:

```
The time is      1000 ns.
```

- If the quotes are omitted:

```
echo The time is    $now ns.
```

all blank spaces of two or more are compressed into one space.

```
The time is $now ns."
```

- echo can also use command substitution, such as:

```
echo The hex value of counter is [examine -hex counter].
```

If the current value of counter is 21 (15 hex), this command returns:

```
The hex value of counter is 15.
```


edit

This command invokes the editor specified by the EDITOR environment variable. By default, the specified filename will open in the Source window.

Syntax

```
edit [<filename>]
```

Arguments

- `<filename>`
(optional) Specifies the name of the file to edit. If the `<filename>` argument is omitted, the editor opens the current source file. If you specify a non-existent filename, it will open a new file. Either absolute or relative paths may be used.

Related Topics

- [notepad](#)
- [EDITOR](#) environment variable

enablebp

This command turns on breakpoints and [when](#) commands that were previously disabled.

Syntax

```
enablebp [<id#> | <label>]
```

Arguments

- [<id#>](#)
(optional) Specifies a breakpoint ID number or when statement to enable.
Note that C breakpoint id numbers have a "c" prefix.
- [<label>](#)
(optional) Specifies the label name of a breakpoint or when statement to enable.

If you do not specify either of these arguments, all breakpoints are enabled.

Use the `bp` command with no arguments to find labels and ID numbers for all breakpoints in the current simulation. Use the `when` command with no arguments to find labels and ID numbers of all when statements in the current simulation.

Related Topics

- [bd](#)
- [bp](#)
- [C Debug](#)
- [disablebp](#)
- [onbreak](#)
- [resume](#)
- [when](#)

enable_menu

This command enables a previously disabled menu. The menu will be changed from grayed-out to normal and will become responsive. Returns nothing.

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

```
enable_menu <window_name> <menu_path>
```

Arguments

- <window_name>
(required) Tk path of the window containing the menu.

Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

- <menu_path>
(required) Name of the Tk menu-widget path.

Examples

- Enable the previously-disabled File menu of the Main window.

```
enable_menu "" File
```
- Enable the previously-disabled File menu of the mywindow window.

```
enable_menu .mywindow File
```

Related Topics

- [add_menu](#)
- [disable_menu](#)

enable_menuitem

This command enables a previously disabled menu item.

The menu item changes from grayed-out to normal, and becomes responsive. Returns nothing.

Arguments to this command are order dependent. Follow the order specified in the Syntax section

Syntax

```
enable_menuitem <window_name> <menu_path> <label>
```

Arguments

- <window_name>
(required) Tk path of the window containing the menu.
Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.
- <menu_path>
(required) Name of the Tk menu-widget path. The path may include a submenu as shown in the example below.
- <label>
(required) Menu item text.

Examples

This command locates the mywindow window and enables the previously-disabled Save Results As... menu item in the save submenu of the file menu.

```
enable_menuitem .mywindow file.save "Save Results As..."
```

Related Topics

- [add_menuitem](#)
- [disable_menuitem](#)

encoding

This command translates between the 16-bit Unicode characters used in Tcl strings and a named encoding, such as Shift-JIS. There are four encoding commands used to work with the encoding of your character representations in the GUI.

- `encoding convertfrom` — Convert a string from the named encoding to Unicode.
- `encoding convertto` — Convert a string to the named encoding from Unicode.
- `encoding names` — Returns a list of all valid encoding names.
- `encoding system` — Changes the current system encoding to a named encoding. If a new encoding is omitted the command returns the current system encoding. The system encoding is used whenever Tcl passes strings to system calls.

Syntax

```
encoding convertfrom <encoding_name> <string>
```

```
encoding convertto <encoding_name> <string>
```

```
encoding names
```

```
encoding system <encoding_name>
```

Arguments

- `string` — Specifies a string to be converted.
- `encoding_name` — The name of the encoding to use.

environment

This command has two forms, `environment` and `env`. It allows you to display or change the current dataset and region/signal environment.

Syntax

`environment [-dataset | -nodataset] [<pathname> | -forward | -back]`

Arguments

- `-dataset`
(optional) Displays the specified environment pathname with a dataset prefix. Dataset prefixes are displayed by default.
- `-nodataset`
(optional) Displays the specified environment pathname without a dataset prefix.
- `<pathname>`
(optional) Specifies a new pathname for the region/signal environment.
If omitted the command causes the pathname of the current region/signal environment to be displayed.
- `-forward`
(optional) Displays the next environment in your history of visited environments.
- `-back`
(optional) Displays the previous environment in your history of visited environments.

Examples

- Display the pathname of the current region/signal environment.
`env`
- Change to another dataset but retain the currently selected context.
`env test:`
- Change all unlocked windows to the context "test:/top/foo".
`env test:/top/foo`
- Move down two levels in the design hierarchy.
`env blk1/u2`
- Move to the top level of the design hierarchy.
`env /`

Related Topics

- See [Object Name Syntax](#) for information on specifying pathnames.
- See [Setting your Context by Navigating Source Files](#) for more information about -forward and -back.

examine

This command has two forms, `examine` and `exa`. It examines one or more objects and displays current values (or the values at a specified previous time) in the Transcript window.

It can also compute the value of an expression of one or more objects.

If the design is being optimized with `vopt`, some of the objects listed below may not be available for viewing. See "[Preserving Object Visibility for Debugging Purposes](#)" for more information.

If you are using C Debug, `examine` can display the value of a C variable as well.

The following objects can be examined:

- **VHDL** — signals, shared variables, process variables, constants, and generics
- **Verilog** — nets, registers, parameters, and variables
- **C** — variables
- **SystemC** — signals, FIFOs, ports, and member variables of modules

When stopped in C code, `examine` (with no arguments) displays the values of the local variables and arguments of the current C function. For specific information related to viewing SystemC objects refer to "[SystemC Object and Type Display](#)".

To display a previous value, specify the desired time using the `-time` option.

To compute an expression, use the `-expr` option. The `-expr` and the `-time` options may be used together.

Virtual signals and functions may also be examined within the GUI (actual signals are examined in the kernel).

The following rules are used by the `examine` command to locate an HDL object:

- If the name does not include a dataset name, then the current dataset is used.
- If the name does not start with a path separator, then the current context is used.
- If the name is a path separator followed by a name that is not the name of a top-level design unit, then the first top-level design unit in the design is used.
- For a relative name containing a hierarchical path, if the first object name cannot be found in the current context, then an upward search is done up to the top of the design hierarchy to look for a matching object name.
- If no objects of the specified name can be found in the specified context, then an upward search is done to look for a matching object in any visible enclosing scope up to an instance boundary. If at least one match is found within a given context, no (more) upward searching is done; therefore, some objects that may be visible from a given context will not be found when wildcards are used if they are within a higher enclosing scope.

- The wildcards '*' and '?' can be used at any level of a name except in the dataset name and inside of a slice specification.
- A wildcard character will never match a path separator. For example, */dut/** will match */dut/signa* and */dut/clk*. However, */dut** won't match either of those.

See [Design Object Names](#) for more information on specifying names.

Syntax

```
examine <name>... [-delta <delta>] [-env <path>] [-event <time>] [-handle] {[ -in] [-out]
[-inout] | [-ports]} [-internal] [-maxlen <integer>] [-expr <expression>] [-name]
[-<radix_type>] [-radix <type>] [-radixenumnumeric | -radixenumsymbolic] [-time <time>]
[-value]
```

Arguments

- <name>...
(required except when specifying -expr.) Specifies the name of any HDL or SystemC object. All object types are allowed, except those of the type file. Multiple names and wildcards are accepted. Spaces, square brackets, and extended identifiers require curly braces; see examples below for more details. To examine a VHDL variable you can add a process label to the name. For example, (make certain to use two underscore characters):
exa line__36/i
- -delta <delta>
(optional) Specifies a simulation cycle at the specified time step from which to fetch the value, where the default is to use the last delta of the time step. You must log the objects to be examined using the [add list](#), [add wave](#), or [log](#) command for the examine command to be able return a value for a requested delta.
<delta> — Any non-negative integer.
- -env <path>
(optional) Specifies a path in which to look for an object name.
<path> — The specified path to a object.
- -event <time>
(optional) Specifies a simulation cycle at the specified event time from which to fetch the value. The event <time> refers to the event time relative to events for all signals in the objects dataset at the specified time. You must log the objects to be examined using the [add list](#), [add wave](#), or [log](#) command for the examine command to be able return a value for a requested event.
- -expr <expression>
(optional) Specifies an expression to be examined. You must log the expression using the [add list](#), [add wave](#), or [log](#) command for the examine command to return a value for a specified expression. The expression is evaluated at the current time simulation. If you also

specify the `-time` argument, the expression will be evaluated at the specified time. It is not necessary to specify `<name>` when using this argument. See [GUI_expression_format](#) for the format of the expression.

`<expression>` — Specifies an expression enclosed in braces (`{}`).

- `-handle`

(optional) Returns the memory address of the specified `<name>`. You can use this value as a tag when analyzing the simulation. This value also appears as the title of a box in the Watch window. This option will not return any value if you are in `-view` mode.

- `-in`

(optional) Specifies that `<name>` include ports of mode IN.

- `-out`

(optional) Specifies that `<name>` include ports of mode OUT.

- `-inout`

(optional) Specifies that `<name>` include ports of mode INOUT.

- `-internal`

(optional) Specifies that `<name>` include internal (non-port) signals.

- `-maxlen <integer>`

(optional) Specifies the maximum number of characters in the output of the command.

`<integer>` — Any non-negative integer where 0 is unlimited.

- `-ports`

(optional) Specifies that `<name>` include all ports. Has the same effect as specifying `-in`, `-inout`, and `-out` together.

- `-name`

(optional) Displays object name(s) and value(s). Related switch is `-value`.

The `lecho` command will return the output of an `examine` command in "pretty-print" format. For example,

```
lecho [examine -name clk prw pstrb]
```

- `-<radix_type>`

(optional) Specifies the radix type for the objects that follow in the command. Valid entries (or any unique abbreviations) are: binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default.

If no radix is specified for an enumerated type, the default radix is used. You can change the default radix for the current simulation using the `radix` command. You can change the default radix permanently by editing the `DefaultRadix` variable in the `modelsim.ini` file.

- **-radix <type>**
(optional) Specifies a user-defined radix. The **-radix <type>** switch can be used in place of the **-<radix_type>** switch. For example, **-radix hexadecimal** is the same as **-hex**.
<type> — binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default.
This option overrides the global setting of the default radix (the **DefaultRadix** variable in the *modelsim.ini* file).
- **-radixenumnumeric**
(optional) Causes Verilog and SystemC enums to be displayed as numbers (formatted by the current radix). This overrides the default behavior of always showing enums symbolically.
- **-radixenumsymbolic**
(optional) Restores the default behavior of displaying Verilog and SystemC enums as symbols by reversing the action of the **-radixenumnumeric** option.
- **-time <time>**
(optional) Specifies the time value between 0 and \$now for which to examine the objects.
<time> — A non negative integer where the default unit is the current time unit. If the **<time>** field uses a unit other than the current unit, the value and unit must be placed in curly braces. For example, the following are equivalent for ps resolution:

```
exa -time {3.6 ns} signal_a  
exa -time 3600 signal_a
```


If an expression is specified it will be evaluated at that time. The objects to be examined must be logged via the add list, add wave, or log command in order for the examine command to be able to return a value for a requested time.
- **-value**
(default) Returns value(s) as a curly-braces separated Tcl list. Use to toggle off a previous use of **-name**.

Examples

- Return the value of */top/bus1*.

```
examine /top/bus1
```
- Return the value of the subelement of *rega* that is specified by the index (i.e., 16). Note that you must use curly braces when examining subelements.

```
examine {rega[16]}
```
- Return the value of the contiguous subelements of *foo* specified by the slice (i.e., 20:22). Note the curly braces.

```
examine {foo[20:22]}
```
- Note that when specifying an object that contains an extended identifier as the last part of the name, there must be a space after the closing `'\` and before the closing `}'`.

examine {/top/My extended id\ }

- In this example, the **-expr** option specifies a signal path and user-defined Tcl variable. The expression will be evaluated at 3450us.

examine -time {3450 us} -expr {/top/bus and \$bit_mask}

- Using the `${fifo}` syntax limits the variable to the simple name `fifo`, instead of interpreting the parenthesis as part of the variable. Quotes are needed when spaces are involved; and by using quotes (“ ”) instead of braces, the Tcl interpreter will expand variables before calling the command.

examine -time \$t -name \$fifo "\${fifo}(1 to 3)" \${fifo}(1)

- Because **-time** is not specified, this expression will be evaluated at the current simulation time. Note the signal attribute and array constant specified in the expression.

examine -expr {clk'event && (/top/xyz == 16'hffae)}

Commands like [find](#) and `examine` return their results as a Tcl list (just a blank-separated list of strings). You can do things like:

```
foreach sig [find sig ABC*] {echo "Signal $sig is [exa $sig]" ...}
```

```
if {[examine -bin signal_12] == "11101111XXXZ"} {...}
```

```
examine -hex [find *]
```

The Tcl variable array, `$examine ()`, can also be used to return values. For example, `$examine (/clk)`. You can also examine an object in the [Source Window](#) by selecting it with the right mouse button.

- Print the value of C variable `x`.

```
examine x
```

- Print the value `*p` (de-references `p`).

```
examine *p
```

- Print the structure member `in1` pointed to by `ip`.

```
examine ip->in1
```

Related Topics

- [Design Object Names](#)
- [Wildcard Characters](#)
- [DefaultRadix](#) `modelsim.ini` variable
- [GUI_expression_format](#)
- [C Debug](#)
- ["Preserving Object Visibility for Debugging Purposes"](#)

exit

This command exits the simulator and the ModelSim application.

If you want to stop the simulation using a [when](#) command, use a [stop](#) command within your when statement, do not use an exit command or a [quit](#) command. The stop command acts like a breakpoint at the time it is evaluated.

Syntax

```
exit [-force] [-code <integer>]
```

Argument

- -force
(optional) Quits without asking for confirmation. If this argument is omitted, ModelSim asks you for confirmation before exiting. You can also use `-f` as an alias for this switch.
- -code <integer>
(optional) Quits the simulation and issues an exit code.
`<integer>` — This is the value of the exit code. You should not specify an exit code that already exists in the tool. Refer to the section "[Exit Codes](#)" in the User's Manual for a list of existing exit codes. You can also specify a variable in place of `<integer>`.

You should always print a message before executing the exit `-code` command to explicitly state the reason for exiting.

Examples

You can use `exit -code` to instruct a [vmake](#) command to exit when it encounters an assertion error. The [onbreak](#) command can specify commands to be executed upon an assert failure of sufficient severity, after which the simulator can be made to return an exit status. This is shown in the following example:

```
set broken 0
onbreak {
    set broken 88
    resume
}
run -all
if { $broken } {
    puts "failure -- exit status $broken"
    exit -code $broken
} else {
    puts "success"
}
quit -f
```

The [resume](#) command gives control back to the commands following the `run -all` to handle the condition appropriately.

fcover configure

This command enables, disables, and sets coverage targets for SystemVerilog and PSL cover directives.

If you invoke it without any optional arguments, the command reports the current settings for the specified directive(s).

Syntax

```
fcover configure <coverage_directive>... [-at_least <count>] [-disable] [-enable] [-exclude] [-include] [-limit <count>] [-log on | off] [-recursive] [-weight <integer>]
```

Arguments

- <coverage_directive>...
(required) Identifies the functional coverage directive(s) to which the configuration parameters should be applied. Multiple directives may be specified. Wildcards are allowed.
- -at_least <count>
(optional) Specifies a target count for the selected coverage point(s).

<count> — Any positive integer where the default value is 1.

A directive is considered 100% covered when the specified count is reached. You can change the permanent default for this argument by editing the [CoverAtLeast](#) variable in the *modelsim.ini* file.
- -disable
(optional) Disables incrementing on the specified directive(s). Disabled directives still count toward overall coverage if they had coverage events prior to being disabled. You can change the permanent default for this argument by editing the [CoverEnable](#) variable in the *modelsim.ini* file.
- -enable
(optional) Enables incrementing on the specified directive(s). You can change the permanent default for this argument by editing the [CoverEnable](#) variable in the *modelsim.ini* file. (default)
- -exclude
(optional) Removes the specified directive(s) from the current functional coverage database. Excluded directives will still show up in the Cover Directives window but they do not count toward coverage totals nor do they show up in reports.
- -include
(optional) Adds the specified directive(s) to the current functional coverage database. (default)

- -limit <count>

(optional) Specifies the number of SystemVerilog or PSL cover directive hits before the directive is auto disabled.

<count> — Any positive integer greater than or equal to the value of -at_least, where the default is -1 (unlimited).

The value for -limit is automatically adjusted to equal the value for -at_least if the value for -at_least is set to a greater value than the value for -limit. You can change the permanent default for this argument by editing the [CoverLimit](#) variable in the *modelsim.ini* file.

This argument is useful if you have a directive that is already covered but is evaluated frequently. Disabling such directives improves simulation performance.

- -log on | off

(optional) Specifies whether to log directive counts to the transcript. Even with logging off, coverage counts are incremented in the underlying database. This option applies to all directives that follow it in the command line.

One of the following values is required:

- on — Enable logging. (default)
- off — Disable logging.

You can change the permanent default for this argument by setting the [CoverLog](#) variable in the *modelsim.ini* file.

- -recursive

(optional) For use with wildcard matching. Specifies that the path of the matching search is to descend recursively into subregions. If omitted, the search is limited to the selected region.

- -weight <integer>

(optional) Specifies a relative weighting for the specified coverage directive.

<integer> — Any natural integer (≥ 0) where the default is 1.

A directive with a weight of 2 will have twice the impact on the aggregated coverage numbers than a directive with a weight of 1. If its weight is set to 0 but a directive is otherwise enabled, its coverage will not count toward the aggregated coverage statistics.

Examples

- Report the current configuration settings for the functional coverage directives in the current scope.

```
fcover configure *
```

- Enable coverage counting for */top/reset_trigger* with an -at_least count of 20.

```
fcover configure -enable -at_least 20 /top/reset_trigger
```

- Disable all functional coverage directives in the region */top* and all regions below */top*.

fcover configure -r -disable /top/*

Related Topics

- [Changing the Default Configuration of Cover Directives](#)
- [CoverAtLeast](#)
- [CoverEnable](#)
- [CoverLimit](#)
- [CoverLog](#)
- [Verification with Functional Coverage](#)
- [Verification with Assertions and Cover Directives](#)

find

This command locates objects by type and name. Arguments to the command are grouped by object type:

- [Arguments for nets and signals](#)
- [Arguments for instances and blocks](#)
- [Arguments for virtuals](#)
- [Arguments for classes](#)
- [Arguments for objects](#)

Syntax

```
find nets | signals <object_name> ... [-internal] [-nofilter] {[-in] [-inout] [-out] | [-ports]}  
[-recursive]
```

```
find instances | blocks {<object_name> ... | -bydu <design_unit> ...} [-recursive] [-nodu]
```

```
find virtuals <object_name> ... [-kind <kind>] [-unsaved] [-recursive]
```

```
find classes [<class_name>]
```

```
find objects [-class <class_name>] [-isa <class_name>] [<object_name>]
```

Arguments for nets and signals

When searching for nets and signals, the find command returns the full pathname of all nets, signals, registers, variables, and named events that match the name specification.

- <object_name> ...
(required) Specifies the net or signal for which you want to search. Multiple nets and signals and wildcard characters are allowed. Wildcards cannot be used inside of a slice specification. Spaces, square brackets, and extended identifiers require special syntax; see the examples below for more details.
- -in
(optional) Specifies that the scope of the search is to include ports of mode IN.
- -inout
(optional) Specifies that the scope of the search is to include ports of mode INOUT.
- -internal
(optional) Specifies that the scope of the search is to include internal (non-port) objects.
- -nofilter
(optional) Specifies that the *WildcardFilter* Tcl preference variable be ignored when finding signals or nets.

find

- -out
(optional) Specifies that the scope of the search is to include ports of mode OUT.
- -ports
(optional) Specifies that the scope of the search is to include all ports. Has the same effect as specifying -in, -out, and -inout together.
- -recursive
(optional) Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region.

Arguments for instances and blocks

When searching for instances, the find command returns the primary design unit name.

- -bydu <design_unit> ...
(required if <object_name> is not specified.) Searches for a design unit.

 <design_unit> ... — Name of a design unit to search for. Multiple design units and wildcard characters are allowed. This argument matches the pattern specified by <design_unit> of the instance, which must be in the form: Library.Primary[Secondary]. The Secondary name is present only for design units that have secondary names, such as VHDL. The Library name is the physical name for the library.
- <object_name> ...
(required if -bydu is not specified.) Specifies the name of an instance or block for which you want to search. Multiple instances and wildcard characters are allowed.
- -recursive
(optional) Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region.
- -nodu
(optional) Removes the "du" string from the names of design units found with -bydu argument.

Arguments for virtuals

When searching for virtuals, all optional arguments must be specified before any object names.

- <object_name> ...
(required) Specifies the virtual object for which you want to search. Multiple virtuals and wildcard characters are allowed.
- -kind <kind>
(optional) Specifies the kind of virtual object for which you want to search.

 <kind> — A virtual object of one of the following kinds:
 - designs

- `explicit`s
 - `function`s
 - `implicit`s
 - `signal`s.
- `-unsaved`
Specifies that ModelSim `find` only virtuals that have not been saved to a format file.

Arguments for classes

- `<class_name>`
(optional) Specifies the incrTcl class for which you want to search. Wildcard characters are allowed. The options for `class_name` include `nets`, `objects`, `signals`, and `virtuals`. If you do not specify a class name, the command returns all classes in the current namespace context. See incrTcl commands in the Tcl Man Pages (Help > Tcl Man Pages) for more information.

Arguments for objects

- `-class <class_name>`
(optional) Restricts the search to objects whose most-specific class is `class_name`.
- `-isa <class_name>`
(optional) Restricts the search to those objects that have `class_name` anywhere in their heritage.
- `<object_name>`
(optional) Specifies the incrTcl object for which you want to search. Wildcard characters are allowed. If you do not specify an object name, the command returns all objects in the current namespace context. See incrTcl commands in the Tcl Man Pages (Help > Tcl Man Pages) for more information.

Description

The following rules are used by the `find` command to locate an object:

- If the name does not include a dataset name, then the current dataset is used.
- If the name does not start with a path separator, then the current context is used.
- If the name is a path separator followed by a name that is not the name of a top-level design unit, then the first top-level design unit in the design is used.
- For a relative name containing a hierarchical path, if the first object name cannot be found in the current context, then an upward search is done up to the top of the design hierarchy to look for a matching object name.
- If no objects of the specified name can be found in the specified context, then an upward search is done to look for a matching object in any visible enclosing scope up to an

instance boundary. If at least one match is found within a given context, no (more) upward searching is done; therefore, some objects that may be visible from a given context will not be found when wildcards are used if they are within a higher enclosing scope.

- The wildcards '*' and '?' can be used at any level of a name except in the dataset name and inside of a slice specification. Square bracket ([]) wildcards can also be used.
- A wildcard character will never match a path separator. For example, `/dut/*` will match `/dut/signa` and `/dut/clk`. However, `/dut*` won't match either of those.
- Because square brackets are wildcards in the find command, only parentheses (()) can be used to index or slice arrays.
- The `WildcardFilter` Tcl preference variable is used by the find command to exclude the specified types of objects when performing the search.

See [Design Object Names](#) for more information on specifying names.

Examples

- Find all signals in the entire design.
find signals -r /*
- Find all input signals in region `/top` that begin with the letters "xy".
find nets -in /top/xy*
- Find all signals in the design hierarchy at or below the region `<current_context>/u1/u2` whose names begin with "cl".
find signals -r u1/u2/cl*
- Find a signal named `s1`. Note that you must enclose the object in curly braces because of the square bracket wildcard characters.
find signals {s[1]}
- Find signals `s1`, `s2`, or `s3`.
find signals {s[123]}
- Find the element of signal `s` that is indexed by the value 1. Note that the find command uses parentheses (()), not square brackets ([]), to specify a subelement index.
find signals s(1)
- Find a 4-bit array named `data`. Note that you must use curly braces ({}) due to the spaces in the array slice specification.
find signals {/top/data(3 downto 0)}
- Note that when specifying an object that contains an extended identifier as the last part of the name, there must be a space after the closing '\' and before the closing '}'.

find signals {/top/My extended id\ }

- If `/dut/core/pclk` exists, prints the message "pclk does exist" in the transcript. This would typically be run in a Tcl script.

```
if {[find signals /dut/core/pclk] != ""} {  
    echo "pclk does exist"  
}
```

- Find instances based on their names using wildcards. Send search results to a text file that lists instance names, including the hierarchy path, on separate lines.

```
# Search for all instances with ul in path  
set pattern_match "*ul*" ;  
  
# Get the list of instance paths  
set inst_list [find instances -r *] ;  
  
# Initialize an empty list to strip off the architecture names  
set  ilist [list] ;  
  
foreach inst $inst_list {  
    set ipath [lindex $inst 0]  
    if {[string match $pattern_match $ipath]} {  
        lappend ilist $ipath  
    }  
}  
# At this point, ilist contains the list of instances only--  
# no architecture names  
#  
# Begin sorting list  
set  ilist [lsort -dictionary $ilist]  
  
# Open a file to write out the list  
set fhandle [open "instancelist.txt" w]  
foreach inst $ilist {  
    # Print instance path, one per line  
    puts $fhandle $inst  
}  
  
# Close the file, done.  
close $fhandle ;
```

Additional search options

To search for HDL objects within a specific display window, use the [search](#) command or select **Edit > Find**.

Related Topics

- [Design Object Names](#)
- [Wildcard Characters](#)

find drivers

This command traces backward in time to find the active driver(s), processes, or first elements of the specified signal or signal event. Processes may be combinatorial or sequential assignments. The command can trace through multiple clock cycles and multiple clock domains. Traces are executed either during simulation or post-simulation.

You can do the following types of causality traces:

- Show Cause — Trace back to the first sequential element causing the event. Duplicates the GUI menu paths: **Show Cause > Show Cause** or **Show Cause > Show Cause from Time**. (See [Initiating Causality Traceback from the GUI](#).)
- Show Driver — Trace the specified signal back to the immediate driving process(es). The process(es) may be combinatorial or sequential assignments. Duplicates the GUI menu paths: **Show Cause > Show Driver** or **Show Cause > Show Driver from Time**.
- Show Root Cause — Trace the event back as far as possible. This may cross multiple clock cycles and multiple clock domains. Duplicates the GUI menu paths: **Show Cause > Show Root Cause** or **Show Cause > Show Root Cause from Time**.
- Show All Possible Drivers — Display all possible driving assignments for a signal (does not use the simulation results to determine.) Duplicates the GUI menu path: **Show Cause > Show All Possible Drivers**.

Note



Traces may find more than one active driver for the specified signal and time. To continue backtracing, you must start a new search on one of the drivers identified in the Multiple Drivers dialog or the transcript. You must also specify the time that the signal was found. Refer to the Example section for more information.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

A help menu is also available by entering **find drivers -help** or **find drivers -?**

Prerequisites

- After optimizing with **vopt**, use the **vdbg** command with the optimized filename; then use the **-debugdb** switch with the **vsim** command.

Or,

- If **vopt** is not used, simulate the design with the **-debugdb** and **-voptargs="+acc"** arguments to the **vsim** command.

Note

The Show Cause and Show Root Cause options will display a warning if you simulate with the `-novopt` switch. You will only have access to the Show Driver option in the Source Window. Full causality traceback functionality requires optimization of your design with `vopt` or `vsim -voptargs`. Refer to [Usage Flow for Causality Traceback](#) for more information.

Syntax**Show driver**

```
find drivers [-schematic] [-last] [-source] [-transcript [-compact <string> | -tcl]
             [-noclip | -width <n>]] [-time <time> [unit]] [-wave] <signal>
```

Show cause

```
find drivers -cause [-schematic] [-last] [-source] [-transcript [-compact <string> | -tcl]
                   [-noclip | -width <n>]] [-time <time> [unit]] [-wave] <signal>
```

Show root cause

```
find drivers -rootcause [-schematic] [-last] [-source] [-transcript [-compact <string> | -tcl]
                       [-noclip | -width <n>]] [-time <time> [unit]] [-wave] <signal>
```

Show all possible drivers

```
find drivers -possible [-schematic] [-last] [-transcript [-compact <string> | -tcl]
                     [-noclip | -width <n>]] [-wave] <signal>
```

Arguments

- `-cause`
(optional) Traces the specified `<signal>` back to the first sequential element causing the event.
- `-compact <string>`
(optional) Specifies a compact format for data output, using a specified text string to separate the fields.
`<string>` — Any combination of characters. Special characters are allowed except a backslash (`\`).
- `-schematic`
(optional) All signals contained in the path found by the trace are placed in a dedicated Schematic window.
- `-last`
(optional) Returns the results from the last completed trace to the transcript. Useful for trying the various format options. Allows you to quickly see how each format option (**`-compact`**, **`-tcl`**, **`-width`**, and **`-noclip`**) affects the output.

- **-noclip**
(optional) Allows columns to be arbitrarily long when returned to the transcript.
- **-possible**
(optional) Display all of the possible driving assignments for a signal (simulation results are not used). The data returned is hyperlinked to the driving signal in the source file.
- **-rootcause**
(optional) Trace the event back as far as possible. This may cross multiple clock cycles and multiple clock domains.
- **-source**
(optional) Opens the source file containing the line of code found by the trace and scrolls to show that line as well as highlight the driving signal. If the trace type is **-possible** then this option is not allowed.
- **-tcl**
(optional) Displays trace results in a TCL list.
- **-time <time> [unit]**
(optional) Traces **<signal>** from a specified time. If not specified, the trace begins from the current active time indicated by either the end of the simulation, or the Wave window's active cursor. The **-time** option is not allowed when using the **-possible** argument.
 - <time>** — Specified as an integer or decimal number. Current simulation units are the default unless specifying **<unit>**.
 - <unit>** — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid VHDL time units are: fs, ps, ns, us, ms, sec, min, and hr. You must enclose **<time>** and **<unit>** within curly braces ({}).
- **-transcript**
(optional) Specifies that trace results are reported to the Transcript window in tabular format (unless the **-compact** argument is used). Default behavior if not using **-schematic**, **-source**, or **-wave**. Returns the following information:
 - “Causality trace from the time **<time>** for the signal **<signal>**”
 - **Time** — Simulation time when the driving signal caused a change in the traced signal.
 - **Type** — Register containing the traced signal.
 - **Scope** — Name of the signal that caused the change in the signal being traced.
 - **Source File** — Location of the signal in the source file. Includes the full path, name of the source file, and line number. The text is hyperlinked to the location of the driving signal. Double-clicking the entry opens the source and highlights the source code.

- `-wave`
(optional) Specifies that all signals in the path found by the trace are added to a dedicated Wave window, and cursors are added that show the beginning and ending times of the trace. The dedicated Wave window is cleared of signals before displaying the results of a new trace.
- `-width <n>`
(optional) Specifies the maximum size of each column when data is returned to the transcript in tabular form.
`<n>` — Any positive integer.
- `<signal>`
(required) The path name of a signal in the Wave window. Must be specified as the final argument to the **find drivers** command.

Example

- Show the first sequential process for the signal `test1/jr`

find drivers -source -time {698 ns} -cause -transcript sim:/test1/jr

```

VSIM 29> find drivers -source -time {698 ns} -cause -transcript sim:/test1/jr
# Causality trace from time 698 ns for signal "sim:/test1/jr":
#
# Time | Type | Scope      | Signal  | Source File
# ---- | ---- | - - - - - | - - - - | - - - - -
# 30 ns | FF   | /test1/ff1 | data    | C:\trace/test1.sv:106
# 25 ns | Gate | /test1     | dbgTemp3 | C:\trace/test1.sv:30
# 25 ns | Gate | /test1     | ar      | C:\trace/test1.sv:30
# 25 ns | Gate | /test1     |         | C:\trace/test1.sv:65

```

Locate the driving signal for the Flip Flop `/test1/ff1` in the source file by double clicking on the linked text:

`C:\trace/test1.sv:106.`

The driving signal is highlighted in the source file.

- Trace a signal that finds Multiple Active Drivers.

find drivers -source -time {867 ns} -cause -transcript sim:/test1/mnf

Returns:

```

# Multiple active drivers exist at time 822 ns for signal
sim:/test1/mnf:
#   PROCESS: #ASSIGN#33  SIGNAL: /test1/ar  FILE:
C:\ctraceback/test1.sv:33

```

```
# PROCESS: #ASSIGN#33 SIGNAL: /test1/br FILE:  
C:\ctraceback/test1.sv:33
```

To continue the trace you must select one of the multiple drivers and execute the `find drivers` command again on that signal, for example:

```
find drivers -source -time {822 ns} -cause -transcript /test1/br
```

Returns:

```
# Causality trace from time 822 ns for signal "sim:/test1/br":  
#  
# Time | Type | Scope | Signal | Source File  
# -----|-----|-----|-----|-----  
# 822 ns | Gate | /test1 |         | C:\ctraceback/test1.sv:66
```

Related Topics

- [vdbg](#)
- [Using Causality Traceback](#)

find infiles

This command searches for a string in the specified file(s) and prints the results to the Transcript window. The results are individually hotlinked and will open the file and display the location of the string.

When you execute this command in command-line mode from outside of the GUI, the results are sent to stdout with no hotlinks.

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

```
find infiles <string_pattern> <file>...
```

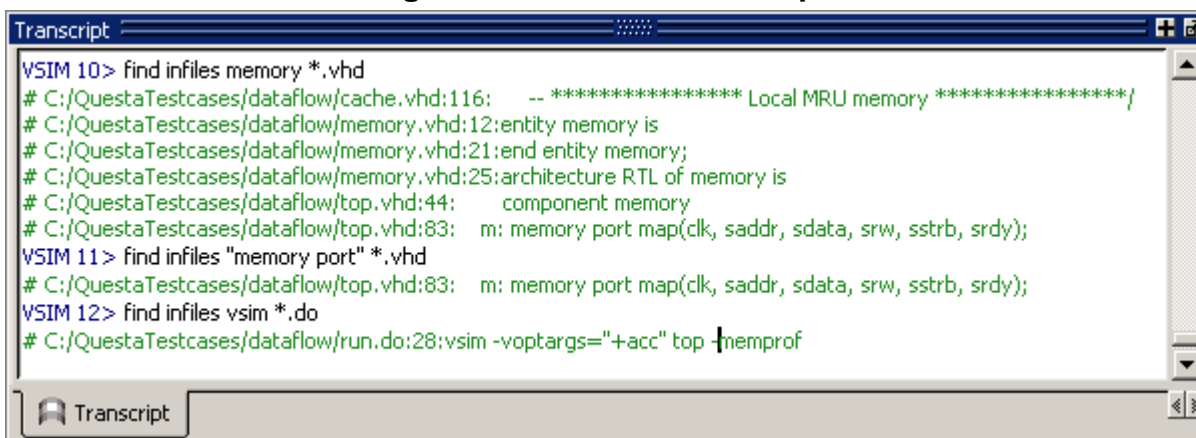
Arguments

- <string_pattern>
(required) The string you are searching for. You can use Tcl regular expression wildcards to further restrict the search capability.
- <file>...
(required) The file(s) to search. You can use Tcl regular expression wildcards to further restrict the search capability.

Example

Figure 2-2 shows a screen capture containing a few examples of the find infiles command and the results.

Figure 2-1. find infiles Example



```
Transcript
VSIM 10> find infiles memory *.vhd
# C:/QuestaTestcases/dataflow/cache.vhd:116:  -- ***** Local MRU memory *****}
# C:/QuestaTestcases/dataflow/memory.vhd:12:entity memory is
# C:/QuestaTestcases/dataflow/memory.vhd:21:end entity memory;
# C:/QuestaTestcases/dataflow/memory.vhd:25:architecture RTL of memory is
# C:/QuestaTestcases/dataflow/top.vhd:44:  component memory
# C:/QuestaTestcases/dataflow/top.vhd:83:  m: memory port map(clk, saddr, sdata, srw, sstrb, srdy);
VSIM 11> find infiles "memory port" *.vhd
# C:/QuestaTestcases/dataflow/top.vhd:83:  m: memory port map(clk, saddr, sdata, srw, sstrb, srdy);
VSIM 12> find infiles vsim *.do
# C:/QuestaTestcases/dataflow/run.do:28:vsim -voptargs="+acc" top memprof
```

find insource

This command searches for a string in the source files for the current design and prints the results to the Transcript window. The results are hotlinked individually and will open the file and display the location of the string.

When you execute this command in command-line mode from outside of the GUI, the results are sent to stdout with no hotlinks.

Syntax

```
find insource <pattern> [-exact | -glob | -regex] [-inline] [-nocase]
```

Arguments

- <pattern>
(required) The string you are searching for. You can use regular expression wildcards to further restrict the search. You must enclose <pattern> in quotes (“ ”) if it includes spaces. You must specify the <pattern> at the end of the command line; any switches specified after <pattern> will not be registered.
- -exact | -glob | -regex
(optional) Defines the style of regular expression used in the <pattern>
 - exact — Indicates that no characters have special meaning, thus disabling wildcard features.
 - glob — (default) Allows glob-style wildcard characters. For more information refer to the Tcl documentation:
Help > Tcl Man Pages
Select “Tcl Commands”, then “string”, then “string match”
 - regex — Allows Tcl regular expressions. For more information refer to the Tcl documentation:
Help > Tcl Man Pages
Select “Tcl Commands”, then “re_syntax”.
- -inline
(optional) Returns the matches in the form of a Tcl list, which disables the hotlink feature but allows for easier post-processing.
- -nocase
(optional) Treats <pattern> as case-insensitive.

Example

Figure 2-2 shows a couple of examples of the find insource command and the results.

Figure 2-2. find insource Example

The screenshot shows a transcript window titled "Transcript" with the following text:

```

VSIM 5> find insource memory
# cache.vhd:116:  -- ***** Local MRU memory *****/
# memory.vhd:12:entity memory is
# memory.vhd:21:end entity memory;
# memory.vhd:25:architecture RTL of memory is
# top.vhd:44:    component memory
# top.vhd:83:    m: memory port map(clk, saddr, sdata, srw, sstrb, srdy);
VSIM 6> find insource "memory port"
# top.vhd:83:    m: memory port map(clk, saddr, sdata, srw, sstrb, srdy);

VSIM 7>

```

The transcript window includes a scroll bar on the right and a "Transcript" tab at the bottom left.

formatTime

This command provides global format control for all time values displayed in the GUI. When specified without arguments, this command returns the current state of the three arguments.

Syntax

```
formatTime [[+|-]commas] [[+|-]nodefunits] [[+|-]bestunits]
```

Arguments

- [+|-]commas
(optional) Insert commas into the time value.
 - + prefix — On
 - prefix — Off. (default)
- [+|-]nodefunits
(optional) Do not include default unit in the time.
 - + prefix — On
 - prefix — Off. (default)
- [+|-]bestunits
(optional) Use the largest unit value possible.
 - + prefix — On
 - prefix — Off. (default)

Examples

- Display commas in time values.
formatTime +commas
Instead of displaying 6458131 ps, the GUI will display 6,458,131 ps.
- Use largest unit value possible.
formatTime +bestunits
Displays 8 us instead of 8,000 ns.

force

This command allows you to apply stimulus interactively to VHDL signals and Verilog nets. When executed without arguments, this command returns a list of the most recently applied force commands and a list of forces coming from the Signal Spy `signal_force()` and `$signal_force()` calls from within VHDL, Verilog, and SystemC source code.

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

It is possible to create a complex sequence of stimuli when the force command is included in a macro file.

This command provides additional information with the `-help` switch.

There are a number of constraints on what you can and cannot force:

- You can force “[Virtual Signals](#)” if the number of bits corresponds to the signal value. You cannot force virtual functions.
- You can force bit-selects or an entire register, you cannot force slices of a register
- You can force signals within SystemC modules, with the following limitations:
 - Only mixed language boundary types are supported.
 - The `-drive` option to force is not supported.
 - Individual bits and slices may not be forced or unforced.
- You cannot force VHDL variables. See the [change](#) command for information on working with VHDL variables.
- In VHDL and mixed models, you cannot force an input port that is mapped at a higher level. In other words, you can force the signal at the top of the hierarchy connected to the input port but you cannot force the input port directly.
- You cannot force a VHDL alias of a VHDL signal.
- You cannot force an input port that has a conversion function on the input.

Syntax

```
force {<object_name> <value>} [-freeze | -drive | -deposit] [-cancel [ @<time>[<unit>]]  
[-repeat [ @<time>[<unit>]] [[ @<time>[<unit>]]] ]...
```

Arguments

- `<object_name>`
(required) Specifies the name of the HDL object to be forced. A wildcard is permitted only if it matches one object. See [Design Object Names](#) for the full syntax of an object name. The object name must specify a scalar type or a one-dimensional array of character enumeration. You may also specify a record subelement, an indexed array, or a sliced array, as long as the type is one of the above. Must be specified as the first argument to the **force** command.

- <value>

(required) Specifies the value to which the object is to be forced. The specified value must be appropriate for the type. Must be specified as the second argument to the **force** command.

A VHDL one-dimensional array of character enumeration can be forced as a sequence of character literals or as a based number with a radix of 2, 8, 10 or 16. For example, the following values are equivalent for a signal of type `bit_vector (0 to 3)`:

Value	Description
1111	character literal sequence
2#1111	binary radix
10#15	decimal radix
16#F	hexadecimal radix

Note



For based numbers in VHDL, ModelSim translates each 1 or 0 to the appropriate value for the number's enumerated type. The translation is controlled by the translation table in the *pref.tcl* file. If ModelSim cannot find a translation for 0 or 1, it uses the left bound of the signal type (`type'left`) for that value.

- -freeze

(optional) Freezes the object at the specified <value> until it is forced again or until it is unforced with the `noforce` command.

- -drive

(optional) Attaches a driver to the object and drives the specified <value> until the object is forced again or until it is unforced with the `noforce` command.

This option is illegal for unresolved signals or SystemC signals.

- -deposit

(optional) Sets the object to the specified <value>. The <value> remains until the object is forced again, there is a subsequent driver transaction, or it is unforced with a `noforce` command.

Note



If the -freeze, -drive, or -deposit options are not used, then -freeze is the default for unresolved objects, and -drive is the default for resolved objects. If you prefer -freeze as the default for resolved and unresolved VHDL signals, change the `DefaultForceKind` variable in the *modelsim.ini* file.

- -cancel [@]<time>[<unit>]

(optional) Cancels the force command at the specified <time>.

@ — A prefix applied to <time> to specify an absolute time interval where the default is to specify a relative time interval by omitting the @ character.

<time> — The time (either relative or absolute) at which to cancel the force command. Any non-negative integer. A value of zero cancels the force at the end of the current time period.

<unit> — A suffix specifying a time unit where the default is to specify the current time unit by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr.

```
-cancel 520          \\ Relative Time
-cancel 520ns       \\ Relative Time
```

Enclose with curly braces ({}) when using spaces between the arguments. See the example below.

```
-cancel {@ 520 ns}  \\ Absolute Time
```

- -repeat [@]<time>[<unit>]

(optional) Repeats the force command. A repeating force command will force a value before other non-repeating force commands that occur in the same time step.

@ — A prefix applied to <time> to specify an absolute time interval where the default is to specify a relative time interval by omitting the @ character.

<time> — The time (either relative or absolute) at which to repeat the force command. Any non-negative integer. A value of zero cancels the force at the end of the current time period. Cancellation occurs at the last simulation delta cycle of a time unit.

<unit> — A suffix specifying a time unit where the default is to specify the current time unit by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr.

Enclose with curly braces ({}) when using spaces between the arguments.

- [@]<time>[<unit>]

(optional) Specifies the time to which the <value> is to be applied.

@ — A prefix applied to <time> to specify an absolute time interval where the default is to specify a relative time interval by omitting the @ character.

<time> — The time (either relative or absolute) to apply to <value>. Any non-negative integer. A value of zero cancels the force at the end of the current time period. Cancellation occurs at the last simulation delta cycle of a time unit.

<unit> — A suffix specifying a time unit where the default is to specify the current time unit by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr.

Enclose with curly braces ({}) when using spaces between the arguments.

Examples

- Force *input1* to 0 at the current simulator time.

```
force input1 0
```

- Force *bus1* to 01XZ at 100 nanoseconds after the current simulator time.

force bus1 01XZ 100 ns

- Force *bus1* to 16#F at the absolute time 200 measured in the resolution units selected at simulation start-up.

force bus1 16#f @200

- Force *input1* to 1 at 10 time units after the current simulation time and to 0 at 20 time units after the current simulation time. This cycle repeats starting at 100 time units after the current simulation time, so the next transition is to 1 at 100 time units after the current simulation time.

force input1 1 10, 0 20 -r 100

- Similar to the previous example, but also specifies the time units. Time unit expressions preceding the "-r" must be placed in curly braces since a space is used between the time value and time unit.

force input1 1 10 ns, 0 {20 ns} -r 100ns

- Force signal *s* to alternate between values 1 and 0 every 100 time units until time 1000. Cancellation occurs at the last simulation delta cycle of a time unit.

force s 1 0, 0 100 -repeat 200 -cancel 1000

So,

force s 1 0 -cancel 0

will force signal *s* to 1 for the duration of the current time period.

- Force *sigA* to decimal value 85 whenever the value on the signal is 1.

```
when {/mydut/siga = 10#1} {  
  force -deposit /mydut/siga 10#85  
}
```

Related Topics

- [change](#)
- [DefaultForceKind](#)
- [Design Object Names](#)
- [Force Command Defaults](#)
- [noforce](#)
- [Virtual Signals](#)

fsm list

This command returns information about recognized finite state machines, including the number of states and transitions.

The output matches that of the FSM List window.

Syntax

```
fsm list
```

Examples

fsm list

Returns:

```
{sim:/top/A/I0/present_state 10 165} {sim:/top/A/I1/present_state 7 98}  
{sim:/top/B/present_state 3 18} {sim:/top/B/I0/present_state 10 165}  
{sim:/top/B/I1/present_state 14 302}
```

which follows the form:

```
{<dataset>:<instance> <number_of_states> <number_of_transitions>}
```

Related Topics

- [fsm view](#)
- [fsm properties](#)
- [Finite State Machines](#)
- [FSM List Window](#)
- [FSM Viewer Window](#)

fsm properties

This command returns information about the specified finite state machine.

This information matches that found in the FSM Properties dialog box.

Syntax

```
fsm properties <fsm_instance>
```

Arguments

- <fsm_instance>
(required) You can produce a list of available fsm instances with the fsm list command.

Examples

```
fsm properties /top/A/I0/present_state
```

Returns:

```
top/A/I0/clock {RESET 0 S0 1 S1 2 S2 3 S3 4 S4 5 S5 6 S6 7 S7 8 S8 9} RESET  
{top/A/I0/#ALWAYS#53 top/A/I0/#ALWAYS#60} {top/A/I0/clock top/A/I0/rst  
top/A/I0/next_state top/A/I0/enable} {top/A/I0/present_state top/A/I0/en2  
top/A/I0/en1 top/A/I0/en0 top/A/I0/dval top/A/I0/fifo_rd}
```

which follows the form:

```
<clock> {<state_encoding>} <reset_state> {process_list} {<clock> <inputs>  
<outputs>}
```

Related Topics

- [fsm view](#)
- [fsm list](#)
- [Finite State Machines](#)
- [FSM List Window](#)
- [FSM Viewer Window](#)

fsm view

This command displays the specified finite state machine in the FSM List window.

Syntax

```
fsm view <fsm_instance>
```

Arguments

- <fsm_instance>
(required) You can produce a list of available fsm instances with the fsm list command.

Examples

```
fsm view /top/A/I0/present_state
```

Returns:

```
{sim:/top/A/I0/present_state 10 165} {sim:/top/A/I1/present_state 7 98}  
{sim:/top/B/present_state 3 18} {sim:/top/B/I0/present_state 10 165}  
{sim:/top/B/I1/present_state 14 302}
```

which follows the form:

```
{<dataset>:<instance> <number_of_states> <number_of_transitions>}
```

Related Topics

- [fsm list](#)
- [fsm properties](#)
- [Finite State Machines](#)
- [FSM List Window](#)
- [FSM Viewer Window](#)

gdb dir

This command sets the source directory search path for the C debugger and starts the C debugger if it is not already running.

Syntax

```
gdb dir [<src_directory_path_1>] ...
```

Argument

- `<src_directory_path_1>...`
(optional) Specifies one or more directories for C source code. If no directory is specified, the source directory search path is set to the gdb default—`$cdir:$cwd`. Either absolute or relative paths may be used. Specify multiple paths as a space separated list. Wildcards and relative paths are allowed.

Examples

Set the source directory search paths to `../dut/` and `../foo/`.

```
gdb dir ../dut/ ../foo/
```

Related Topics

- [C Debug](#)
- [Setting Up C Debug](#)

getactivecursortime

This command gets the time of the active cursor in the Wave window and returns the time value.

Syntax

```
getactivecursortime [-window <wname>]
```

Arguments

- `-window <wname>`
(optional) Specifies an instance of the Wave window that is not the default.
`<wname>` — The name of the window that is not the default.
Use the [view](#) command to change the default window.

Examples

```
getactivecursortime
```

Returns:

```
980 ns
```

Related Topics

- [left](#)
- [right](#)

getactivemarkertime

This command gets the time of the active marker in the List window.

Returns the time value. If -delta is specified, returns time and delta.

Syntax

```
getactivemarkertime [-window <wname>] [-delta]
```

Arguments

- -window <wname>
(optional) Specifies an instance of the List window that is not the default. Otherwise, the default List window is used.

 <wname> — The name of the window that is not the default.

 Use the [view](#) command to change the default window.
- -delta
(optional) Returns the delta value where the default is to return only the time.

Examples

```
getactivemarkertime -delta
```

Returns:

```
980 ns, delta 0
```

Related Topics

- [down](#)
- [up](#)

help

This command displays in the Transcript window a brief description and syntax for the specified command.

Syntax

```
help [<command> | <topic>]
```

Arguments

- <command>
(optional) Specifies the command for which you want help. The entry is case and space sensitive.
- <topic>
(optional) Specifies a topic for which you want help. The entry is case and space sensitive. Specify one of the following six topics:

Topic	Description
commands	Lists all available commands and topics
debugging	Lists debugging commands
execution	Lists commands that control execution of your simulation.
Tcl	Lists all available Tcl commands.
Tk	Lists all available Tk commands
incrTCL	Lists all available incrTCL commands

history

This command lists the commands you have executed during the current session. History is a Tcl command. For more information, consult the Tcl Man Pages (Help > Tcl Man Pages).

Syntax

```
history [clear] [keep <value>]
```

Arguments

- clear
(optional) Clears the history buffer.
- keep <value>
(optional) Specifies the number of executed commands to keep in the history buffer.
<value> — Any positive integer where the default is 50.

jobspy

This command controls JobSpy, a tool for monitoring and controlling batch simulations and simulation farms.

This command provides additional information with the `-help` switch.

Syntax

```
jobspy [-gui | -startd | -killd | jobs | status | <jobid> <command>]
```

Arguments

- `-gui`
(optional) Launches the JobSpy Job Manager GUI.
- `-startd`
(optional) Starts the jobspy daemon which enables job tracking. You must first set the `JOBSPY_DAEMON` environment variable before starting the daemon. Refer to “[Starting the JobSpy Daemon](#)” for further details.
- `-killd`
(optional) Terminates the JobSpy daemon.
- `jobs`
(optional) Returns a list of current jobs with a variety of status information (e.g., job ID, current simulation time, start time, etc.).
- `status`
(optional) Returns the status of the JobSpy daemon.
- `<jobid> <command>`
(optional) Specifies a job ID to be processed by `<command>`. Use `jobspy jobs` to get a list of job IDs.
`<command>` — A JobSpy simulator command. Refer to “[Simulation Commands Available to JobSpy](#)” for a list of valid commands.

Related Topics

- [Monitoring Simulations with JobSpy](#)
- [Running the JobSpy GUI](#)
- [Simulation Commands Available to JobSpy](#)
- [Starting the JobSpy Daemon](#)

layout

This command allows you to perform a number of editing operations on custom GUI layouts, such as loading, saving, maximizing, and deleting.

The command options include:

- layout active – returns the current active window
- layout current – lists the current layout
- layout delete – removes the current layout from the .modelsim file (UNIX/Linux) or Registry (Windows)
- layout load – opens the specified layout
- layout names – lists all known layouts
- layout normal – minimizes the current maximized window
- layout maximized – return a 1 if the current layout is maximized, or a 0 if minimized
- layout save – saves the current layout to the specified name
- layout togglezoom – toggles the current zoom state of the active window (from minimized to maximized or maximized to minimized)
- layout zoomactive – maximizes the current active window
- layout zoomwindow – maximizes the specified window

Syntax

layout active

layout current

layout delete <**name**>

layout load <**name**>

layout names

layout normal

layout maximized

layout save <**name**>

layout togglezoom

layout zoomactive

layout zoomwindow <**window**>

Arguments

- <name>
(required) Specifies the name of the layout.
- <window>
(required) The window specification can be any format accepted by the [view](#) command. The window can be specified by its type (i.e., wave, list, objects, etc.), by the windowobj name (i.e., .main_pane.wave, .main_pain.library, etc.), or by the tab name (i.e., wave1, list3, etc.)

Related Topics

- [Customizing the Simulator GUI Layout](#)

lecho

This command takes one or more Tcl lists as arguments and pretty-prints them to the Transcript window.

Syntax

```
lecho <args> ...
```

Arguments

- <args> ...
Any Tcl list created by a command or user procedure. Specified as a space separated list.

Examples

- Print the Wave window configuration list to the Transcript window.

```
lecho [configure wave]
```

left

This command searches left (previous) for signal transitions or values in the specified Wave window.

It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions, to find the time at which a waveform takes on a particular value, or to find an expression of multiple signals that evaluates to true. See the [right](#) command for related functionality.

The procedure for using left entails three steps:

1. Click on the desired waveform.
2. Click on the desired starting location. (The [seetime](#) command can initially position the cursor from the command line, if desired.)
3. Issue the left command.

Returns: <number_found> <new_time> <new_delta>

Syntax

```
left [-expr {<expression>}] [-falling] [<n>] [-rising] [-value <sig_value> [-noglitch]]  
    [-window <wname>]
```

Arguments

- -expr {<expression>}
(optional) The waveform display is searched until the expression evaluates to a boolean true condition.

 <expression> — An expression that involves one or more objects, but limited to objects that have been logged in the referenced waveform display. An object may be specified either by its full path or by the shortcut label displayed in the Wave window.

See [GUI_expression_format](#) for the format of the expression. The expression must be placed within curly braces ({}).
- -falling
(optional) Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored.
- <n>
(optional) Specifies to find the nth match where the default is 1. If less than n are found, the number found is returned with a warning message, and the cursor is positioned at the last match.

left

- **-noglitch**
(optional) Looks at signal values only on the last delta of a time step. For use with the **-value** option only.
- **-rising**
(optional) Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored.
- **-value <sig_value>**
(optional) Specifies the value of the object to match.

 <sig_value> — A value specified in the same radix that the selected object is displayed. Case is ignored, but otherwise the value must be an exact string match. Don't-care bits are not supported. Only one signal can be selected, but that signal may be an array.
- **-window <wname>**
(optional) Specifies an instance of the Wave window that is not the default. When <wname> is not specified, the default Wave window is used. Use the [view](#) command to change the default window.

 <wname> — The name of a Wave window not currently the default.

Examples

- Find the second time to the left at which the selected vector transitions to FF23, ignoring glitches.

 left -noglitch -value FF23 2
- Go to the previous transition on the selected signal.

 left

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the [GUI_expression_format](#).

- Search left for an expression that evaluates to a boolean 1 when signal *clk* just changed from low to high and signal *mystate* is the enumeration reading and signal */top/u3/addr* is equal to the specified 32-bit hex constant; otherwise is 0.

 left -expr {clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)}
- Search left for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal */top/u3/addr* equals hex ac.

 left -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
- Search left for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, clock just changed from low to high, and signal *mode* is enumeration writing.

 left -expr {{{(NOW > 23 us) && (NOW < 54 us)}} && clk'rising && (mode == writing)}

Note

“[Wave Window Mouse and Keyboard Shortcuts](#)” are also available for next and previous edge searches. Tab searches right (next) and shift-tab searches left (previous).

Related Topics

- [GUI_expression_format](#)
- [right](#)
- [seetime](#)
- [view](#)

log

This command creates a wave log format (WLF) file containing simulation data for all HDL objects whose names match the provided specifications. Objects that are displayed using the [add list](#) and [add wave](#) commands are automatically recorded in the WLF file. By default the file is named *vsim.wlf* and stored in the current working directory. You can change the default name using the `-wlf` option of the [vsim](#) command or by setting the [WLFfilename](#) variable in the *modelsim.ini* file.

If no port mode is specified, the WLF file contains data for all objects in the selected region whose names match the object name specification.

The WLF file contains a record of all data generated for the list and wave windows during a simulation run. Reloading the WLF file restores all objects and waveforms and their complete history from the start of the logged simulation run. See [dataset open](#) for more information.

For all transaction streams created through the SCV or Verilog APIs, logging is enabled by default. A transaction is logged to the WLF file if logging is enabled at the beginning of a simulation run when the design calls `::begin_transaction()` or `$begin_transaction`. The effective start time of the transaction (the time passed by the design as a parameter to `::begin_transaction`) is irrelevant. For example, a stream could have logging disabled between T1 and T2 and still record a transaction in that period, through retroactive logging after time T2. A transaction is always either entirely logged or entirely ignored.

Transaction streams created from an MVC (Multiview Verification Component) are not logged by default: they must be logged explicitly through the [log](#), [add wave](#), or [add list](#) commands.

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Note



The log command is also known as the "add log" command.

Syntax

```
log [-class <classtype>] [-flush] [-howmany] {[-in] [-inout] [-out] | [-ports]}
    [-internal] [-mvcall] [-mvcovm] [-optcells] [-out] [-ports] [-recursive [-depth <level>]]
    <object_name> ...
```

Arguments

- `-class <classtype>`

(optional) Log all objects of a class specified in `<classtype>` that are generated after the command is invoked. Descends the hierarchy recursively to include all properties of `<classtype>` that are also classes.

`<classtype>` — The type of class to be logged.

Caution

Using this switch can result in a large amount of logged data.

- **-depth <level>**
(optional) Restricts a recursive search (specified with the `-recursive` argument) to a certain level of hierarchy.
`<level>` — Any non-negative integer. For example, if you specify `-depth 1`, the command descends only one level in the hierarchy.
- **-flush**
(optional) Adds region data to the WLF file after each individual log command. Default is to add region data to the log file for the following conditions:
 - A command is executed that advances simulation time (e.g., `run`, `step`, etc.).
 - You quit the simulation.
- **-howmany**
(optional) Returns an integer indicating the number of signals found.
- **-in**
(optional) Specifies that the WLF file is to include data for ports of mode IN whose names match the specification.
- **-inout**
(optional) Specifies that the WLF file is to include data for ports of mode INOUT whose names match the specification.
- **-internal**
(optional) Specifies that the WLF file is to include data for internal (non-port) objects whose names match the specification.
- **-mvcall**
(optional) Specifies the inclusion of all MVC protocol transactions when a wildcard is used (i.e. `log -r /*`). By default, MVC transactions are sometimes excluded from logging with normal wildcard usage.
- **-mvcovm**
(optional) Specifies the inclusion of all OVM sequence transactions when a wildcard is used (i.e. `log -r /*`). By default, OVM transactions are sometimes excluded from logging with normal wildcard usage.

log

- **-optcells**
(optional) Makes Verilog optimized cell ports visible when using wildcards. By default Verilog optimized cell ports are not selected even if they match the specified wildcard pattern.
- **-out**
(optional) Specifies that the WLF file is to include data for ports of mode OUT whose names match the specification.
- **-ports**
(optional) Specifies that the scope of the search is to include all ports, IN, INOUT, and OUT.
- **-recursive**
(optional) Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region. You can use the **-depth** argument to specify how far down the hierarchy to descend.
- **<object_name>**
(required) Specifies the object name that you want to log. Must be specified as the final argument to the **log** command. Multiple object names are specified as a space separated list. Wildcard characters are allowed. Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.

By default, wildcard card logging does not log the internals of cells. Refer to the [+libcell](#) argument of the **vlog** command for more information.

Examples

- Log all objects in the design.
log -r /*
- Log all output ports in the current design unit.
log -out *

Related Topics

- [add list](#)
- [add wave](#)
- [dataset alias](#)
- [dataset clear](#)
- [dataset close](#)
- [dataset config](#)
- [dataset config](#)
- [dataset info](#)
- [dataset list](#)
- [dataset open](#)
- [dataset rename](#)
- [dataset restart](#)
- [dataset save](#)
- [dataset snapshot](#)
- [nolog](#)
- [Recording Simulation Results With Datasets](#)
- [vlog +libcell](#)
- [What is an MVC?](#)
- [Wildcard Characters](#)

lshift

This command takes a Tcl list as an argument and shifts it in-place, one place to the left, eliminating the left-most element.

The number of shift places may also be specified. Returns nothing.

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

`lshift <list> [<amount>]`

Arguments

- `<list>`
(required) Specifies the Tcl list to target with `lshift`. Must be specified as the first argument to the **lshift** command.
- `<amount>`
(optional) Specifies the number of places to shift where the default is 1.

Examples

```
proc myfunc args {  
    # throws away the first two arguments  
    lshift args 2  
    ...  
}
```

Related Topics

- See the Tcl man pages (Help > Tcl Man Pages) for details.

lsublist

This command returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern.

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

```
lsublist <list> <pattern>
```

Arguments

- <list>
(required) Specifies the Tcl list to target with lsublist.
- <pattern>
(required) Specifies the pattern to match within the <list> using Tcl glob-style matching.

Examples

- In the example below, variable 't' returns "structure signals source".

```
set window_names "structure signals variables process source wave  
list"  
set t [lsublist $window_names s*]
```

Related Topics

- The set command is a Tcl command. See the Tcl man pages (Help > Tcl Man Pages) for details.

mem compare

This command compares a selected memory to a reference memory or file. Must have the "diff" utility installed and visible in your search path in order to run this command.

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

```
mem compare {[-mem <ref_mem>] | [-file <ref_file>]} [actual_mem]
```

Arguments

- -mem <ref_mem>
(optional) Specifies a reference memory to be compared with actual_mem.
 <ref_mem> — A memory record.
- -file <ref_file>
(optional) Specifies a reference file to be compared with actual_mem.
 <ref_file> — A saved memory file.
- actual_mem
(required) Specifies the name of the memory to be compared against the reference data.
Must be specified as the final argument to the **mem compare** command.

mem display

This command prints to the Transcript window the memory contents of the specified instance. If the given instance path contains only a single array signal or variable, the signal or variable name need not be specified.

You can redirect the output of the mem display command into a file for later use with the mem load command. The output file can also be read by the Verilog \$readmem system tasks if the memory module is a Verilog module and Verilog memory format (hex or binary) is specified.

Address radix, data radix, and address range for the output can also be specified, as well as special output formats.

By default, identical data lines are printed. To replace identical lines with a single line containing the asterisk character, you can enable compression with the -compress argument.

Note



The format settings are stored at the top of this file as a pseudo comment so that subsequent mem load commands can correctly interpret the data. Do not edit this data when manipulating a saved file.

Arguments to this command are order dependent. Please read through the argument descriptions for more information.

Syntax

```
mem display [-addressradix [d | h]] [-compress] [-dataradix <radix_type>]
            [-endaddress <end>][-format [bin | hex | mti]] [-noaddress] [-startaddress <st>]
            [-wordspersline <n>] [<path>]
```

Arguments

- -addressradix [d | h]
(optional) Specifies the address radix for the default (MTI) formatted files.
 - d — Decimal radix. (default if -format is specified as mti.)
 - h — Hex radix.
- -compress
(optional) Specifies that identical lines not be printed. Reduces the file size by replacing exact matches with a single line containing an asterisk. These compressed files are automatically expanded during a mem load operation.
- -dataradix <radix_type>
(optional) Specifies the data radix for the default (MTI) formatted files. If unspecified, the global default radix is used.
 - <radix_type> A specified radix type. Valid entries (or any unique abbreviations) are: binary, decimal, unsigned, octal, hex, symbolic, and default. If no radix is specified for an enumerated type, the symbolic representation is used.

You can change the default radix type for the current simulation using the [radix](#) command or make the default radix permanent by editing the [DefaultRadix](#) variable in the `modelsim.ini` file.

- `-endaddress <end>`
(optional) Specifies the end address for a range of addresses to be displayed.
`<end>` — Any valid address in the memory. If unspecified, the default is the end of the memory.
- `-format [bin | hex | mti]`
(optional) Specifies the output format of the contents.
`bin`— Specifies a binary output.
`hex`— Specifies a hex output.
`mti` — MTI format. (default).
- `-noaddress`
(optional) Specifies that addresses not be printed.
- `-startaddress <st>`
(optional) Specifies the start address for a range of addresses to be displayed.
`<st>` — Any valid address in the memory. If unspecified, the default is the start of the memory.
- `-wordspersline <n>`
(optional) Specifies how many words are to be printed on each line.
`<n>` — Any positive integer where the default is an 80 column display width.
- `<path>`
(required) Specifies the full path to the memory instance. The default is the current context, as shown in the Structure window. Indexes can be specified. Must be specified as the final argument to the **mem display** command.

Examples

- This command displays the memory contents of instance `/top/c/mru_mem`, addresses 5 to 10:

```
mem display -startaddress 5 -endaddress 10 /top/c/mru_mem
```

returns:

```
# 5: 110 110 110 110 110 000
```

- Display the memory contents of the same instance to the screen in hex format, as follows:

```
mem display -format hex -startaddress 5 -endaddress 10 /top/c/mru_mem
```

returns:

5: 6 6 6 6 6 0

Related Topics

- For details on MTI format, see the description contained in [mem load](#).

mem list

This command displays a flattened list of all memory instances in the current or specified context after a design has been elaborated.

Each instance line is prefixed by "VHDL:" or "Verilog:", depending on the type of model.

Returns the signal/variable name, address range, depth, and width of the memory.

Syntax

```
mem list [-r] [<path>]
```

Arguments

- **-r**
(optional) Recursively descends into sub-modules when listing memories.
- **<path>**
(optional) The hierarchical path to the location the search should start where the default is the current context, as shown in the Structure window.

Examples

- Recursively list all memories at the top level of the design.

```
mem list -r /
```

Returns:

```
# Verilog: /top/m/mem[0:255](256d x 16w)  
#
```

- Recursively list all memories in */top2/uut*.

```
mem list /top2/uut -r
```

Returns:

```
# Verilog: /top2/uut/mem[0:255] x 16w
```

mem load

This command updates the simulation memory contents of a specified instance. You can upload contents either from a memory data file, a memory pattern, or both. If both are specified, the pattern is applied only to memory locations not contained in the file.

A relocatable memory file is one that has been saved without address information. You can load a relocatable memory file into the instance of a memory core by specifying an address range on the mem load command line. If no address range (starting and ending address) is specified, the memory is loaded starting at the first location.

The order in which the data is placed into the memory depends on the format specified by the -format option. If you choose bin or hex format, the memory is filled low to high, to be compatible with \$readmem commands. This is in contrast to the default MTI format, which fills the memory according to the memory declaration, from left index to right index.

For Verilog objects and VHDL integers and std_logic types: if the word width in a file is wider than the word width of the memory, the leftmost bits (msb) in the data words are ignored. To allow wide words use the -truncate option which will ignore the msb bits that exceed the memory word size. If the word width in the file is less than the width of the memory, and the leftmost digit of the file data is not 'X', then the leftmost bits are zero filled. Otherwise, they are X-filled.

The type of data required for the -filldata argument is dependent on the -filltype specified: a fixed value, or one that governs an incrementing, decrementing, or random sequence.

- For fixed pattern values, the fill pattern is repeatedly tiled to initialize the memory block specified. The pattern can contain multiple word values for this option.
- For incrementing or decrementing patterns, each memory word is treated as an unsigned quantity, and each successive memory location is filled in with a value one higher or lower than the previous value. The initial value must be specified.
- For a random pattern, a random data sequence will be generated to fill in the memory values. The data type in the sequence will match the type stored in the memory. For std_logic and associated types, unsigned integer sequences are generated. A seed value may be specified on the command line. For any given seed, the generated sequence is identical.

The interpretation of the pattern data is performed according to the default system radix setting. However, this can be overridden with a standard Verilog-style '<radix_char><data>' specification.

Syntax

```
mem load -infile <infile> [-endaddress <end>]  
      [-filltype {dec | inc | rand | value} -filldata <data_word>] [-fillradix <radix_type>]  
      [-format [bin | hex | mti]] [<path>] [-skip <Nwords>] [-startaddress <st>] [-truncate]
```

Arguments

- **-infile <infile>**
(Required unless the **-filltype** argument is used.) Updates memory data from the specified file.
<infile> — The name of a memory file.
- **-endaddress <end>**
(optional) Specifies the end address for a range of addresses to be loaded.
<end>— Specified as any valid address in the memory.
- **-filltype {dec | inc | rand | value }**
(Required unless the **-infile** argument is used, in which case it is optional.) Fills in memory addresses in an algorithmic pattern starting with the data word specified in **-filldata**. If a fill pattern is used without a file option, the entire memory or specified address range is filled with the specified pattern.
dec — Decrement each succeeding memory word by one digit.
inc — Increment each succeeding memory word by one digit.
rand — Randomly generate each succeeding memory word starting with the word specified by **-filldata** as the seed.
value — Value (default) Substitute each memory word in the range with the value specified in **-filldata**.
- **-filldata <data_word>**
(required when **-filltype** is used) Specifies a data word used to fill memory addresses in the pattern specified by **-filltype**.
<data_word> — Specifies a data word. Must be in the same format as specified by the **-fillradix** switch.
- **-fillradix <radix_type>**
Specifies radix of the data specified by the **-filldata** switch.
<radix_type> — Valid entries (or any unique abbreviations) are: binary, decimal, unsigned, octal, hex, symbolic, and default.
- **-format [bin | hex | mti]**
(optional) Specifies the format of the file to be loaded.
bin— Specifies binary data format.
hex— Specifies hex format.
mti — MTI format. (default).
Specifies the format of the file to be loaded. The **bin** and **hex** values are the standard Verilog hex and binary memory pattern file formats. These can be used with Verilog memories, and with VHDL memories composed of **std_logic** types.

In the MTI memory data file format, internal file address and data radix settings are stored within the file itself. Thus, there is no need to specify these settings on the mem load command line. If a format specified on the command line and the format signature stored internally within the file do not agree, the file cannot be loaded.

- <path>
(optional) The hierarchical path to the memory instance. If the memory instance name is unique, shorthand instance names can be used. The default is the current context, as shown in the Structure window.

Memory address indexes can be specified in the instance name also. If addresses are specified both in the instance name and the file, only the intersection of the two address ranges is populated with memory data.
- -skip <Nwords>
(optional) Specifies the number of words to be skipped between each fill pattern value. Used with -filltype and -filldata.

<Nwords> — Specified as an unsigned integer.
- -startaddress <st>
(optional) Specifies the start address for a range of addresses to be loaded.

<st> — Any valid address in the memory.
- -truncate
(optional) Ignores any most significant bits (msb) in a memory word which exceed the memory word size. By default, when memory word size is exceeded, an error results.

Examples

- Load the memory pattern from the file *vals.mem* to the memory instance */top/m/mem*, filling the rest of the memory with the fixed-value 1'b0.

```
mem load -infile vals.mem -format bin -filltype value -filldata 1'b0  
/top/m/mem
```

When you enter the mem display command on memory addresses 0 through 12, you see the following:

```
mem display -startaddress 0 -endaddress 12 /top/m/mem
# 0: 0000000000000000 0000000000000001 0000000000000010 0000000000000011
# 4: 0000000000000100 0000000000000101 0000000000000110 0000000000000111
# 8: 0000000000001000 0000000000001001 0000000000000000 0000000000000000
# 12: 0000000000000000
```

- Load the memory pattern from the file *vals.mem* to the memory instance */top/m/mru_mem*, filling the rest of the memory with the fixed-value 16'Hbeef.

```
mem load -infile vals.mem -format hex -st 0 -end 12 -filltype value -filldata 16'Hbeef  
/top/m/mru_mem
```

- Load memory instance */top/mem2* with two words of memory data using the Verilog Hex format, skipping 3 words after each fill pattern sequence.

mem load -filltype value -filldata "16'hab 16'hcd" /top/mem2 -skip 3

- Truncate the msb bits that exceed the maximum word size (specified in HDL code).

mem load -format h -truncate -infile data_files/data.out /top/m_reg_inc/mem

Related Topics

- [mem save](#)

mem save

This command saves the contents of a memory instance to a file in any of the supported formats: Verilog binary, Verilog hex, and MTI memory pattern data.

This command works identically to the **mem display** command, except that its output is written to a file rather than a display.

The order in which the data is placed into the saved file depends on the format specified by the **-format** argument. If you choose **bin** or **hex** format, the file is populated from low to high, to be compatible with \$readmem commands. This is in contrast to the default **mti** format, which populates the file according to the memory declaration, from left index to right index.

You can use the **mem save** command to generate relocatable memory data files. The **-noaddress** option omits the address information from the memory data file. You can later load the generated memory data file using the **memory load** command.

Syntax

```
mem save -outfile <filename> [-addressradix {dec | hex}] [-dataradix <radix_type>] [-format  
{bin | hex | mti}] [-compress | -noaddress] [<path>]  
[-startaddress <st> -endaddress <end>] [-wordspersline <Nwords>]
```

Arguments

- **-outfile** <filename>
(required) Specifies that the memory contents are to be stored in a file.

<filename> — The name of the file where the specified memory contents are to be stored.
- **-addressradix** {dec | hex}
(optional) Specifies the address radix for the default mti formatted files.

dec — Decimal (default).
hex — Hexadecimal.
- **-compress**
(optional) Specifies that only unique lines are printed, identical lines are not printed. Mutually exclusive with the **-noaddress** switch.
- **-dataradix** <radix_type>
(optional) Specifies the data radix for the default mti formatted files.

<radix_type> — Valid entries (or any unique abbreviations) are: binary, decimal, unsigned, octal, hex, and symbolic.

You can change the default radix for the current simulation using the [radix](#) command. You can change the default radix permanently by editing the [DefaultRadix](#) variable in the modelsim.ini file.

- **-endaddress <end>**
(optional) Specifies the end address for a range of addresses to be saved.
<end> — Any valid address in the memory.
- **-format {bin | hex | mti}**
(optional) Specifies the format of the output file.
bin— Binary data format.
hex— Hexadecimal format.
mti — MTI format. (default).

The **bin** and **hex** values are the standard Verilog hex and binary memory pattern file formats. These can be used with Verilog memories, and with VHDL memories composed of `std_logic` types.

In the MTI memory data file format, internal file address and data radix settings are stored within the file itself.

- **-noaddress**
(optional) Prevents addresses from being printed. Mutually exclusive with the **-compress** switch.
- **<path>**
(optional) The hierarchical path to the location of the memory instance. The default is the current context, as shown in the Structure window.
- **-startaddress <st>**
(optional) Specifies the start address for a range of addresses to be saved.
<st> — Any valid address in the memory.
- **-wordspersline <Nwords>**
(optional) Specifies how many memory values are to be printed on each line.
<Nwords> — Any unsigned integer where the default assumes an 80 character display width.

Examples

- Save the memory contents of the instance `/top/m/mem(0:10)` to `memfile`, written in the mti radix.

mem save -format mti -outfile memfile -start 0 -end 10 /top/m/mem

The contents of `memfile` are as follows:

```
// memory data file (do not edit the following line - required for mem
load use)
// format=mti addressradix=d dataradix=s version = 1.0
0: 0000000000000000 00000000000000001 0000000000000010 0000000000000011
4: 0000000000000100 0000000000000101 0000000000000110 0000000000000111
8: 0000000000001000 0000000000001001 xxxxxxxxxxxxxxxxxxxx
```

Related Topics

- [mem display](#)
- [mem load](#)

mem search

This command finds and prints to the screen the first occurring match of a specified memory pattern in the specified memory instance. Shorthand instance names are accepted.

Optionally, you can instruct the command to print all occurrences. The search pattern can be one word or a sequence of words.

Syntax

```
mem search { -glob <word> [<word>...] | -regexp <word> [<word>...]}  
  [-addressradix {dec | hex}] [-dataradix <radix_type>] [-all] [-replace <word> [<word>...]]  
  [-startaddress <address>] [-endaddress <address>] [<path>]
```

Arguments

- **-glob** <word> [<word>...]

(required unless using **-regexp**) Specifies the value of the pattern, accepting glob pattern syntax for the search.

<word> — Any word pattern. Multiple word patterns are specified as a space separated list. Wildcards are accepted in the pattern.

This argument and **-regexp** are mutually exclusive arguments.

- **-regexp** <word> [<word>...]

(required unless using **-glob**) Specifies the value of the pattern, accepting regular expression syntax for the search.

<word> — Any word pattern. Wildcards are accepted in the pattern. Multiple word patterns are specified as a space separated list.

This argument and **-glob** are mutually exclusive arguments.

- **-addressradix** {dec | hex}

(optional) Specifies the radix for the address being displayed.

dec — Decimal (default).

hex — Hexadecimal.

- **-all**

(optional) Searches the specified memory range and returns all matching occurrences to the transcript. By default only the first matching occurrence is printed.

- **-dataradix** <radix_type>

(optional) Specifies the radix for the memory data being displayed.

<radix_type> — Can be specified as symbolic, binary, octal, decimal, unsigned, or hex. By default the radix displayed is the system default.

You can change the default radix for the current simulation using the `radix` command. You can change the default radix permanently by editing the `DefaultRadix` variable in the `modelsim.ini` file.

- `-endaddress <address>`
(optional) Specifies the end address for a range of addresses to search.
`<address>` — Any valid address in the memory.
- `<path>`
(optional) Specifies the hierarchical path to the location of the memory instance. The default is the current context, as shown in the Structure window.
- `-replace <word> [<word>...]`
(optional) Replaces the found patterns with a designated pattern.
`<word>` — A word pattern Multiple word patterns are accepted, separated by a single space. No wildcards are allowed in the replaced pattern.
- `-startaddress <address>`
(optional) Specifies the start address for a range of addresses to search.
`<address>` — Any valid address in the memory.

Examples

- Search for and print to the screen all occurrences of the pattern **16'Hbeef** in `/uut/u0/mem3`:

```
mem search -glob 16'Hbeef -dataradix hex /uut/u0/mem3
```

Returns:

```
#7845: beef
#7846: beef
#100223: beef
```

- Search for and print only the first occurrence of **16'Hbeef** in the address range 7845:150000, replacing it with **16'Hcafe** in `/uut/u1/mem3`:

```
mem search -glob 16'Hbeef -d hex -replace 16'Hcafe -st 7846 -end 150000
/uut/u1/mem3
```

Returns:

```
#7846: cafe
```

- Replace all occurrences of **16'Hbeef** with **16'Habe** in `/uut/u1/mem3`:

```
mem search -glob 16'Hbeef -r 16'Habe -addressadix hex -all /uut/u1/mem3
```

Returns:

```
#lea5: 2750  
#lea6: 2750  
#1877f: 2750
```

- Search for and print the first occurrence any pattern ending in f:
mem search -glob "*f"
- Search for and print the first occurrence of this multiple word pattern:
mem search -glob "abe cafe" /uut/u1/mem3
- Search for patterns "0000 0000" or "0001 0000" in *m/mem*:
mem search -data hex -regexp {000[0|1] 0{4}} m/mem -all
- Search for a pattern that has any number of 0s followed by any number of 1s as a memory location, and which has a memory location containing digits as the value:
mem search -regexp {^0+1+\$ \d+} m/mem -all
- Search for any initialized location in a VHDL memory:
mem search -regexp {[^U]} -all <vhdl_memory>

messages clearfilter

This command removes any filter you have set in the Message Viewer. Refer to the section “[Message Viewer Filter Dialog Box](#)” for additional information about filtering in the Message Viewer.

Syntax

```
messages clearfilter
```

Arguments

- No arguments

messages setfilter

This command performs the same action as the [Message Viewer Filter Dialog Box](#), which controls which messages are shown in the Message Viewer.

The ideal workflow for using this command is through the GUI:

1. **View > Message Viewer.**
2. Right-click in the Message Viewer and select **Filter.**

The Message Viewer Filter dialog box is displayed

3. Create your filter.
4. **OK** or **Apply.**

The Message Viewer updates based on your filter and a `messages setfilter` command, which is equivalent to your settings, is output to the transcript.

5. Retain the `messages setfilter` command from the transcript for future use.

Syntax

```
messages setfilter <tcl_list>
```

Arguments

- `<tcl_list>` — The `tcl_list` argument is a complex string of tcl code that controls the filter settings.

Examples

- Severity is error and time is greater than or equal to 100 ns

```
messages setfilter {{} \  
  ( Severity Contains {Case Insensitive} error ) } \  
  {AND ( Time >= 100 ns ) }
```

- The objects field contains neither clock or reset

```
messages setfilter {{} \  
  ( Object Contains {Case Sensitive} clock ) } \  
  {NOR ( Object Contains {Case Sensitive} data ) }
```

- The message string either contains `reg_str2` or `reg_str1`

```
messages setfilter {{} \  
  ( Message Contains {Case Insensitive} reg_str2 ) } \  
  {OR ( Message Contains {Case Insensitive} reg_str1 ) }
```

modelsim

The **modelsim** command starts the ModelSim GUI without prompting you to load a design.

This command is valid only for Windows platforms and may be invoked in one of three ways:

- from the DOS prompt
- from a ModelSim shortcut
- from the Windows Start > Run menu

To use **modelsim** arguments with a shortcut, add them to the target line of the properties of that shortcut. (As expected, arguments also work on the DOS command line.)

You can invoke the simulator from either the ModelSim> prompt after the GUI starts or from a DO file called by **modelsim**.

Syntax

```
modelsim [-do <macrofile>] [<license_option>] [-nosplash]
```

Arguments

- -do <macrofile>
(optional) Executes a DO file when **modelsim** is invoked.
 <macrofile> — The name of a DO file

Note



In addition to the macro called by this argument, if a DO file is specified by the STARTUP variable in *modelsim.ini*, it will be called when the [vsim](#) command is invoked.

- <license_option>
(optional) Restricts the search of the license manager.
- -nosplash
(optional) Disables the splash screen.

Related Topics

- [do](#)
- [vsim](#)
- See [vsim \[<license_option>\]](#) for more information on license options.
- [Using a Startup File](#)

next

This command continues a search after you have invoked the **search** command.

Syntax

```
next <window_name> [-window <wname>]
```

Arguments

- <window_name>
(required) Specifies one window in which to continue searching. Can be one of the following windows: Signals, Objects, Variables, Locals, Source, List, Wave, Process, Structure. Unique abbreviations are accepted.
- -window <wname>
(optional) Specifies an instance of the window that is not the default.

<wname> — The name of a window instance that is not the default. Otherwise, the default window is used.

Use the [view](#) command to change the default window.

Related Topics

- [search](#)

noforce

This command removes the effect of any active force commands on the selected HDL objects. and also causes the object's value to be re-evaluated.

You can use noforce on signals within SystemC modules, with the following limitations:

- Only mixed language boundaries types are supported.
- Individual bits and slices may not be forced or unforced.

Syntax

```
noforce <object_name> ...
```

Arguments

- <object_name>
(required) Specifies the name of an object. Must match an object name used in a previous [force](#) command. Multiple object names may be specified as a space separated list. Wildcard characters are allowed.

Related Topics

- [force](#)
- [Wildcard Characters](#)

nolog

This command suspends writing of data to the wave log format (WLF) file for the specified signals.

A flag is written into the WLF file for each signal turned off, and the GUI displays "-No Data-" for the signal(s) until logging (for the signal(s)) is turned back on. Logging can be turned back on by issuing another **log** command or by doing a **nolog -reset**.

Because use of the **nolog** command adds new information to the WLF file, WLF files created when using the **nolog** command cannot be read by older versions of the simulator. If you are using *dumplog64.c*, you will need to get an updated version.

Transactions written in SCV or Verilog are logged automatically, and can be removed with the **nolog** command. A transaction is logged into the *.wlf* file if logging is enabled (in other words, if no **nolog** command has disabled it) for that stream at the time when the transaction was begun. An entire span of a transaction is either logged or not logged, regardless of the begin and end times specified for that transaction.

Syntax

```
nolog [-all] [-depth <level>] [-howmany] [-in] [-inout] [-internal] [-out] [-ports] [-recursive]
      [-reset] [<object_name>...]
```

Arguments

- -all
(optional) Turns off logging for all signals currently logged.
- -depth <level>
(optional) Restricts a recursive search (specified with the **-recursive** argument) to a certain level of hierarchy.

 <level> — An integer greater than or equal to zero. For example, if you specify -depth 1, the command descends only one level in the hierarchy.
- -howmany
(optional) Returns an integer indicating the number of signals found.
- -in
(optional) Turns off logging only for ports of mode IN whose names match the specification.
- -inout
(optional) Turns off logging only for ports of mode INOUT whose names match the specification.
- -internal
(optional) Turns off logging only for internal (non-port) objects whose names match the specification.

- **-out**
(optional) Turns off logging only for ports of mode OUT whose names match the specification.
- **-ports**
(optional) Specifies that the scope of the search is to include all ports.
- **-recursive**
(optional) Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region. You can use the **-depth** argument to specify how many levels of the hierarchy to descend.
- **-reset**
(optional) Turns logging back on for all unlogged signals.
- **<object_name>...**
(optional) Specifies the object name which you want to unlog. Multiple object names may be specified as a space separated list. Wildcard characters are allowed.

Examples

- Unlog all objects in the design.
nolog -r /*
- Turn logging back on for all unlogged signals.
nolog -reset

Related Topics

- [add list](#)
- [add wave](#)
- [log](#)

notepad

This command opens a simple text editor. It may be used to view and edit ASCII files or create new files.

This mode can be changed from the Notepad Edit menu.

Returns nothing.

Syntax

```
notepad [<filename>] [-r | -edit]
```

Arguments

- <filename>
(optional) Name of the file to be displayed.
- -r
(optional) Specifies read-only mode.
- -edit
(optional) Specifies editing mode. Will not save changes to an existing file that has the Read-only attribute turned on. (default)

noview

This command closes a window in the ModelSim GUI. To open a window, use the **view** command.

Syntax

```
noview [<window_name>...]
```

Arguments

- <window_name>...

(required) Specifies the window(s) to close. Multiple window types may be specified in a space separated list. Wildcards permitted. At least one type (or wildcard) is required.

Refer to the view command for a complete list of possible arguments.

You can also close Source windows using the tab or file name.

Examples

- Close the Wave window named "wave1".

```
noview wave1
```

- Close all List windows.

```
noview List
```

Related Topics

- [view](#)

nowhen

This command deactivates selected when commands.

Syntax

```
nowhen [<label>]
```

Arguments

- <label>
(optional) Specifies an individual when command. Wildcards may be used to select more than one when command.

Examples

- Deactivate the [when](#) command labeled 99.

```
nowhen 99
```

- Deactivate all [when](#) commands.

```
nowhen *
```

Related Topics

- [when](#)

onbreak

This command is used within a macro and specifies one or more commands to be executed when running a macro that encounters a breakpoint in the source code. The command must be followed by a **run** command to take effect.

Using the **onbreak** command without arguments will return the current **onbreak** command string. An **onbreak** command can contain macro calls.

Use the **resume** command to resume execution of the macro file after encountering a breakpoint.

Use an empty string to change the **onbreak** command back to its default behavior:

```
onbreak ""
```

In this case, the macro will be interrupted after a breakpoint occurs (after any associated **bp** command string is executed).

Syntax

```
onbreak {[<command>[; <command>] ...]}
```

Arguments

- <command>

(optional) Any command can be used as an argument to **onbreak**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces ({}). You must use the **onbreak** command before a **run**, **run -continue**, or **step** command. It is an error to execute any commands within an **onbreak** command string following any of the **run** commands. This restriction applies to any macros or Tcl procedures used in the **onbreak** command string.

Examples

- Examine the value of the HDL object data when a breakpoint is encountered. Then continue the run command.

```
onbreak {exa data ; cont}
```

- Resume execution of the macro file on encountering a breakpoint.

```
onbreak {resume}
```

- This set of commands test for assertions. Assertions are treated as breakpoints if the severity level is greater than or equal to the current BreakOnAssertion variable setting (refer to [modelsim.ini Variables](#)). By default a severity level of failure or above causes a breakpoint; a severity level of error or below does not.

```
set broken 0
onbreak {
  set broken 1
  resume
}
run -all
if { $broken } {
  puts "failure"
} else {

  puts "success"
}
```

Related Topics

- [abort](#)
- [bp](#)
- [do](#)
- [onerror](#)
- [resume](#)
- [status](#)
- [Useful Commands for Handling Breakpoints and Errors](#)
- [Macros \(DO Files\)](#)

onElabError

This command specifies one or more commands to be executed when an error is encountered during the elaboration portion of a **vsim** command. The command is used by placing it within a macro.

Use the **onElabError** command without arguments to return to a prompt.

Syntax

```
onElabError {[<command> [<command>] ...]}
```

Arguments

- <command>
(optional) Any command can be used as an argument to **onElabError**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces ({}).

Related Topics

- [do](#)

onerror

This command is used within a macro before a **run** command; it specifies one or more commands to be executed when a running macro encounters an error.

Using the **onerror** command without arguments will return the current **onerror** command string. Use an empty string (**onerror** "") to change the **onerror** command back to its default behavior. Use **onerror** with a **resume** command to allow an error message to be printed without halting the execution of the macro file.

You can also set the global `OnErrorDefaultAction` Tcl variable to dictate what action ModelSim takes when an error occurs. To set the variable on a permanent basis, you must define the variable in a *modelsim.tcl* file (Refer to “[The modelsim.tcl File](#)” for details).

When your **onerror** command is successful, the macro will continue normally, unless your command instructs the tool to quit, for example:

```
onerror {quit -f}
```

or

```
onerror {break}
```

However, if your **onerror** command is not successful, the simulator will be halted, for example:

```
onerror {add wave b}
```

when you don't have a signal named b.

The **onerror** command is executed when a Tcl command (for example, **break**.) encounters an error in the macro file that contains the **onerror** command (note that a **run** command does not necessarily need to be in process). Conversely, `OnErrorDefaultAction` will run even if the macro does not contain a local **onerror** command. This can be useful when you run a series of macros from one script, and you want the same behavior across all macros.

Syntax

```
onerror {[<command> [; <command>] ...]}
```

Arguments

- <command>

(optional) Any command can be used as an argument to **onerror**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces ({}).

Example

- Force the simulator to quit if an error is encountered while the macro is running.

```
onerror {quit -f}
```

Related Topics

- [abort](#)
- [do](#)
- [onbreak](#)
- [resume](#)
- [status](#)
- [Useful Commands for Handling Breakpoints and Errors](#)
- [Macros \(DO Files\)](#)

onfinish

This command controls simulator behavior when encountering \$finish or sc_stop() in the design code. When you specify this command without an argument, it returns the current setting.

Syntax

onfinish [ask | exit | final | stop | default]

Arguments

ask — (optional) In batch mode, the simulation will exit; in GUI mode, the user is prompted for action.

exit — (optional) The simulation exits without asking for any confirmation.

final — (optional) The simulation executes all finish blocks before exiting.

stop — (optional) The simulation ends but remains loaded in memory, allowing for easier post-simulation tasks.

default — (optional) Uses the current [OnFinish](#) setting in the *modelsim.ini* file.

Related Topics

- [OnFinish modelsim.ini variable](#)
- [vsim -onfinish](#)
- [sc_stop Behavior](#)

pa msg

This command controls the display of warning and error messages that occur during Power Aware dynamic checks during simulation run time. You can run this command at the VSIM prompt or in a do file.

Syntax

```
pa msg {-enable | -disable} [-severity {note | warning | error | fatal}] [-stopafter <integer>]  
      [-pa_checks <vopt_values>] [-domain <string>] [<message_num> ...]
```

Arguments

- **-enable | -disable**
Specifies whether to display or suppress messages provided by Power Aware dynamic checking during simulation run time (default severity of warning/error). By default, all messages are displayed (-enable).
- **-severity note | warning | error | fatal**
Sets the severity type for one or more messages (specified by message_id).
- **-stopafter <integer>**
Stops the simulation after one or more messages are displayed the specified number of times.
- **-pa_checks={rop | rpo | rcs | rsa | r | iep | idp | irc | it | isa | i | t | cp | p | a | npu | ugc | upc | s | sdl | sil | sml | snl | srl | svl | ul | uil | uml}**
Applies the pa msg command to Power Aware dynamic checks, using the same argument values available for the vopt -pa_checks command.
- **-domain <string>**
Restricts messages to those pertaining to the power domain specified by <string>.
- **<message_num>**
Specifies the identification number of one or more messages you want to suppress, display, or set the severity for (use spaces to separate multiple message numbers). Optional. If you do not provide a message number, -enable and -disable affect all dynamic checking messages.

Examples

- Suppress messages 8908 and 8903 from simulation time 0 to time 50, then allow those messages to be displayed from time 50 to time 100. (By default, all other messages are displayed from time 0 to time 100.)

```
pa msg -disable 8908 8903  
run 50  
pa msg -enable 8908 8903  
run 50
```

pa msg

- Suppress all messages from simulation time 125 to time 275, then allow all messages to be displayed after time 275.

```
run 125
pa msg -disable
run 150
pa msg -enable
run 350
```

- Set the severity of message number 8901 to “warning.”

```
pa msg -severity warning 8901
```

- Enable message 8905, if it was disabled previously, and change its severity to “fatal” (which stops the simulation on first occurrence).

```
pa msg -enable -severity fatal 8905
```

- Stop the simulation if a retention, toggle, or always-on check occurs three times.

```
pa msg -stopafter 3 -pa_checks=r+t+a
```

pause

This command interrupts the execution of a macro and allows you to perform interactive debugging of a macro file. The command is placed within the macro to be debugged.

Syntax

pause

Arguments

- None.

Description

When a macro is interrupted during execution, the macro returns the prompt:

```
VSIM(paused)>
```

This “pause” prompt notifies you that a macro has been interrupted.

When a macro is paused, you can invoke another macro. If the second macro is interrupted, you can continue invoking macros up to a nesting level of 50 macros.

The [status](#) command lists summary information about all interrupted macros.

Use the [resume](#) command to resume execution of the macro. Use the [abort](#) command to stop execution of some or all of the macros.

Related Topics

- [abort](#)
- [do](#)
- [resume](#)
- [run](#)
- [status](#)

pop

This command moves the specified number of call frames up the C callstack.

This command is used with C Debug.

Syntax

```
pop <#_of_levels>
```

Arguments

- `<#_of_levels>`
(optional) Specifies the number of call frames to move up the C callstack. If unspecified, movement is up one level.

Examples

- Move up 1 call frame.
pop
- Move up 4 call frames.
pop 4

Related Topics

- [push](#)
- [C Debug](#)

power add

This command specifies the signals or nets to monitor for power information. When **power add** is called on a signal or net, vsim keeps a record of any toggling activity of that signal. The information is sent a file when you run the [power report](#) command. This data can be translated and used by third-party power analysis tools.

The basic steps for using this command are:

1. Add the signals or nets of interest with the **power add** command.
2. Run the simulation with the [run](#) command.
3. Produce a report with the **power report** command.

Note



You can use the [power off](#) command to disable monitoring between runs and then use the [power on](#) command to resume monitoring.

Arguments to this command are order-dependent. Please read the argument descriptions for more information.

Syntax

```
power add [[-in] [-out] [-inout] | [-ports]] [-internal] [-nocellnet] [-r] <signals_nets> ...
```

Arguments

- -in
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode IN if they match the <signal_net> specification.
- -out
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode OUT if they match the <signal_net> specification.
- -inout
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode INOUT if they match the <signal_net> specification.
- -ports
(optional) *For use with wildcard searches.* Specifies that the scope of the listing is to include ports of modes IN, OUT, or INOUT if they match the <signal_net> specification.
- -internal
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include internal objects (non-port objects) if they match the <signal_net> specification.

power add

- **-nocellnet**
(optional) Prevents the vsim command from monitoring cell-net for toggling or any power-related activity.
- **-r**
(optional) Searches recursively on a wildcard specified for the signal or net.
- **<signals_nets> ...**
(required) Specifies the signal or net path to monitor. Must be specified as the final argument to the **power add** command.

You can specify multiple names and also use wildcards. The signals and nets must refer to:

- VHDL — signals of type bit, std_logic, or std_logic_vector
- Verilog — nets

When using wildcards, the **-in**, **-inout**, **-internal**, **-out**, and **-ports** arguments filter the qualifying signals.

If you specify more than one of the arguments (**-in**, **-inout**, **-internal**, **-out**, or **-ports**), the logical OR of the arguments is performed.

Related Topics

- [power off](#)
- [power on](#)
- [power report](#)
- [power reset](#)

power off

The power off command works in conjunction with the power add command to make vsim stop updating toggle activity data for the specified signal or net.

After this command is executed, every subsequent run command ignores the power add command for the specified arguments to the **power off** command.

Arguments to this command are order-dependent. Please read the argument descriptions for more information.

Syntax

```
power off [-all] [[-in] [-out] [-inout] | [-ports]] [-internal] [-r] <signals_nets> ...
```

Arguments

- -all
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include inputs, inouts, and outputs. if they match the <signal_nets> specification.
- -in
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode IN if they match the <signal_nets> specification.
- -out
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode OUT if they match the <signal_nets> specification.
- -inout
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode INOUT if they match the <signal_nets> specification.
- -ports
(optional) *For use with wildcard searches.* Specifies that the scope of the listing is to include ports of modes IN, OUT, or INOUT if they match the <signal_nets> specification.
- -internal
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include internal objects (non-port objects) if they match the <signal_nets> specification.
- -r
(optional) Searches recursively on a wildcard specified for the signal or net.
- <signals_nets> ...
(required) Specifies the signal or net to monitor. Must be specified as the final argument to the **power off** command.

You can specify multiple names and also use wildcards. The signals and nets must refer to:

- VHDL — signals of type bit, std_logic, or std_logic_vector

- Verilog — nets

When using wildcards, the **-in**, **-inout**, **-internal**, **-out**, and **-ports** arguments filter the qualifying signals.

If you specify more than one of the arguments (**-in**, **-inout**, **-internal**, **-out**, or **-ports**), the logical OR of the arguments is performed.

Example

Assume that signal /top/a toggles every 5ns.

- Without running the power off command:

```
power add /top/a  
run 400ns  
power report
```

Returns:

Node	Tc	Ti	Time At 1	Time At 0	Time At X
/top/a	80	0	200	200	0

- Running the power off command (and resuming with power on):

```
power add /top/a  
run 100ns  
power off  
run 100ns  
power on  
run 200ns  
power report
```

Returns:

Node	Tc	Ti	Time At 1	Time At 0	Time At X
/top/a	60	0	150	150	0

Related Topics

- [power add](#)
- [power on](#)
- [power report](#)
- [power reset](#)

power on

This command works in conjunction with the power add command to make vsim begin or resume updating toggle activity data for the specified signal or net.

After this command is executed, every subsequent run command implements the power add command.

Arguments to this command are order-dependent. Please read the argument descriptions for more information.

Syntax

```
power on [-all] [[-in] [-out] [-inout] | [-ports]] [-internal] [-r] <signals_nets> ...
```

Arguments

- -all
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include inputs, inouts, and outputs. if they match the <signal_nets> specification.
- -in
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode IN if they match the <signal_nets> specification.
- -out
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode OUT if they match the <signal_nets> specification.
- -inout
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode INOUT if they match the <signal_nets> specification.
- -ports
(optional) *For use with wildcard searches.* Specifies that the scope of the listing is to include ports of modes IN, OUT, or INOUT if they match the <signal_nets> specification.
- -internal
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include internal objects (non-port objects) if they match the <signal_nets> specification.
- -r
(optional) Searches recursively on a wildcard specified for the signal or net.
- <signals_nets> ...
(required) Specifies the signal or net to monitor. Must be specified as the final argument to the **power on** command.

You can specify multiple names and also use wildcards. The signals and nets must refer to:

- VHDL — signals of type bit, std_logic, or std_logic_vector

power on

- o Verilog — nets

When using wildcards, the **-in**, **-inout**, **-internal**, **-out**, and **-ports** arguments filter the qualifying signals.

If you specify more than one of the arguments (**-in**, **-inout**, **-internal**, **-out**, or **-ports**), the logical OR of the arguments is performed.

Example

Assume that signal /top/a toggles every 5ns.

- Without running the power off command:

```
power add /top/a
run 400ns
power report
```

Returns:

Node	Tc	Ti	Time At 1	Time At 0	Time At X
/top/a	80	0	200	200	0

- Running the power off command (and resuming with power on):

```
power add /top/a
run 100ns
power off
run 100ns
power on
run 200ns
power report
```

Returns:

Node	Tc	Ti	Time At 1	Time At 0	Time At X
/top/a	60	0	150	150	0

Related Topics

- [power add](#)
- [power off](#)
- [power report](#)
- [power reset](#)

power report

This command reports power information for the objects specified with [power add](#).

The report can be in either a tabular ASCII format (-file) or a Switching Activity Interchange Format (SAIF) format (-bsaif).

Data produced by these commands can be translated and used by third-party tools used for power analysis. The **power report** command is intended to be used as follows:

1. Add the objects of interest with the [power add](#) command.
2. Run the simulation with the [run](#) command.
3. Produce the report with the **power report** command.

Syntax

```
power report [-all] [-noheader] [-file <filename>] [-bsaif <filename>]
```

Arguments

- -all
(optional) Writes information on all objects logged with power add.
If you do not specify this argument, the report lists only those signals or nets that have a toggle count that is non-zero.
- -noheader
(optional) Suppresses the header to aid in post processing.
This argument has no affect on the output from the -bsaif switch.
- -file <filename>
(optional) Specifies an ASCII-format power report is to be created.
<filename> — A user-specified name for the report.
If you do not specify this argument or the **-bsaif** argument, the tabular ASCII-format power report is returned to the transcript. You can specify both **-file** and **-bsaif** on the same command line, resulting in both reports being generated.
- -bsaif <filename>
(optional) Specifies a backward-SAIF format power report.
<filename> — A user-specified name for the report.
If you do not specify this argument or the **-file** argument, the tabular ASCII-format power report is returned to the transcript. You can specify both **-file** and **-bsaif** on the same command line, resulting in both reports being generated.

Description

The report format for each line is:

power report

Node, Tc, Ti, Time at 1, Time at 0, Time at X

- Node — The hierarchical path of the signal, net or port.
- Tc — (toggle count) The number of 0->1 and 1->0 transitions.
- Ti — (hazard count) The number of 0/1->X, and X->0/1 transitions.

Note

Note that if a signal is initialized at X, and later transitions to 0 or 1, it is not counted as a hazard.

- Time at X — The length of time spent at each of the three respective states.

Example Reports

The following example is from the -file output of the tabular ASCII-format power report.

```
#
# Power Report Interval
# 1100000 ps
#
# Power Report      Node      Tc      Ti      Time At 1      Time At 0      Time At X
# -----
# /test_sm/out_wire(7)      1      0      669000 ps      420000 ps      11000 ps
# /test_sm/out_wire(6)      2      0      520000 ps      569000 ps      11000 ps
# /test_sm/out_wire(5)      3      0      149000 ps      940000 ps      11000 ps
# /test_sm/out_wire(4)      2      0      60000 ps      1029000 ps      11000 ps
# /test_sm/out_wire(3)      1      0      669000 ps      420000 ps      11000 ps
# ...
```

The following example is from the -bsaif output of the backward SAIF format power report. This file contains Header information (between SAIFILE and DURATION entries) and specific instance information (INSTANCE entries)

```
(SAIFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN )
(VENDOR "Mentor Graphics")
(PROGRAM_NAME "vsim")
(VERSION "2.3b")
(TIMESCALE 100ps)
(DIVIDER /)
(DURATION 1200000 ps)
(INSTANCE
  (INSTANCE test_sm
    (NET out_wire(7)
      (T0 420000 ps) (T1 769000 ps) (TX 11000 ps)
      (TC 1) (IG 0)
    )
  )
)
(INSTANCE
  (INSTANCE test_sm
    (NET out_wire(6)
      (T0 660000 ps) (T1 529000 ps) (TX 11000 ps)
    )
  )
)
```

```
        (TC 3) (IG 0)
    )
)
...

```

Related Topics

- [power add](#)
- [power off](#)
- [power on](#)
- [power reset](#)

power reset

This command selectively resets power information to zero for the signals or nets specified with the power add command.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
power reset [-all] [[-in] [-out] [-inout] | [-ports]] [-internal] [-r] <signals_nets> ...
```

Arguments

- -all
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include inputs, inouts, and outputs. if they match the <signal_nets> specification.
- -in
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode IN if they match the <signal_nets> specification.
- -out
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode OUT if they match the <signal_nets> specification.
- -inout
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include ports of mode INOUT if they match the <signal_nets> specification.
- -ports
(optional) *For use with wildcard searches.* Specifies that the scope of the listing is to include ports of modes IN, OUT, or INOUT if they match the <signal_nets> specification.
- -internal
(optional) *For use with wildcard searches.* Specifies that the scope of the search is to include internal objects (non-port objects) if they match the <signal_nets> specification.
- -r
(optional) Searches recursively on a wildcard specified for the signal or net.
- <signals_nets> ...
(required) Specifies the signal or net to monitor. Must be specified as the final argument to the **power reset** command.

You can specify multiple names and also use wildcards. The signals and nets must refer to:

- VHDL — signals of type bit, std_logic, or std_logic_vector
- Verilog — nets

When using wildcards, the **-in**, **-inout**, **-internal**, **-out**, and **-ports** arguments filter the qualifying signals.

If you specify more than one of the arguments (**-in**, **-inout**, **-internal**, **-out**, or **-ports**), the logical OR of the arguments is performed.

Related Topics

- [power add](#)
- [power off](#)
- [power on](#)
- [power report](#)

precision

This command determines how real numbers display in the graphic interface (e.g., Objects, Wave, Locals, and List windows). It does not affect the internal representation of a real number and therefore precision values over 17 are not allowed.

Executing the **precision** command without any arguments returns the current precision setting.

Syntax

`precision [<digits>[#]]`

Arguments

- `<digits>[#]`
(optional) Specifies the number of digits to display where the default is 6.
— A suffix that forces the display of trailing zeros. See examples for more details.

Examples

- Results in 4 digits of precision.

precision 4

For example:

1.234 or 6543

- Results in 8 digits of precision including trailing zeros.

precision 8#

For example:

1.2345600 or 6543.2100

- Results in 8 digits of precision but doesn't print trailing zeros.

precision 8

For example:

1.23456 or 6543.21

printenv

This command prints to the Transcript window the current names and values of all environment variables.

If variable names are given as arguments, returns only the names and values of the specified variables.

Syntax

```
printenv [<var>...]
```

Arguments

- <var>...
(optional) Specifies the name(s) of the environment variable(s) to print.

Examples

- Print all environment variable names and their current values.

```
printenv
```

Returns:

```
# CC = gcc  
# DISPLAY = srl:0.0  
...
```

- Print the specified environment variables:

```
printenv USER HOME
```

Returns:

```
# USER = vince  
# HOME = /scratch/srl/vince
```

process report

This command creates a textual report of all processes displayed in the Process Window.

Syntax

```
process report [-file <filename>] [-append]
```

Arguments

- **-file <filename>**
(optional) Creates an external file where raw process data will be saved. If **-file** is not specified, then the output is redirected to stdout.
<filename> — A user-specified name for the file.
- **-append**
(optional) Specifies that process data is to be appended to the current process report file. If this option is not used, the process data will overwrite the existing process report file.

profile clear

This command clears any performance data that has been gathered during previous executions of the [run](#) command. Use the [profile on](#) command to begin profiling.

After this command is executed, all profiling data is reset.

This command has no effect on the current profiling session. The last [profile on](#) or [profile off](#) command will still be in effect.

Prerequisites

First enable profiling with the [profile on](#) command.

Syntax

```
profile clear
```

Arguments

- None

Related Topics

- [profile interval](#)
- [profile off](#)
- [profile on](#)
- [profile open](#)
- [profile option](#)
- [profile reload](#)
- [profile report](#)
- [profile save](#)
- [Profiling Performance and Memory Use](#)

profile interval

This command selects the frequency with which the profiler collects samples during a [run](#) command.

Prerequisites

First enable profiling with the [profile on](#) command.

Syntax

```
profile interval [<sample_frequency>]
```

Arguments

- `<sample_frequency>`

Any integer from 1 to 999 that represents how many milliseconds (ms) to wait between each sample collected during a profiled simulation run where the default is 10 ms.

If the **profile interval** command is executed without arguments, the current sample frequency is returned.

Related Topics

- [profile clear](#)
- [profile off](#)
- [profile on](#)
- [profile open](#)
- [profile option](#)
- [profile reload](#)
- [profile report](#)
- [profile save](#)
- [Profiling Performance and Memory Use](#)

profile off

This command disables runtime memory allocation and statistical performance profiling.

Syntax

```
profile off [-assertions] [-classes] [-cvg] [-qdas] [-solver] [-m] [-p]
```

Arguments

- **-assertions**
(optional) Disables fine-grain analysis of memory capacity data being collected for assertions and cover directives.
- **-classes**
(optional) Disables fine-grain analysis of memory capacity data being collected for class objects.
- **-cvg**
(optional) Disables fine-grain analysis of memory capacity data being collected for covergroups.
- **-qdas**
(optional) Disables fine-grain analysis of memory capacity data being collected for queues, dynamic arrays, associative arrays.
- **-solver**
(optional) Disables fine-grain analysis of memory capacity data being collected for randomize () calls.
- **-m**
(optional) Disables memory allocation profiling only.
- **-p**
(optional) Disables statistical performance profiling only.

Related Topics

- [profile clear](#)
- [profile interval](#)
- [profile on](#)
- [profile open](#)
- [profile option](#)
- [profile reload](#)
- [profile report](#)
- [profile save](#)
- [Profiling Performance and Memory Use](#)

profile on

This command enables runtime memory allocation and statistical performance profiling. After this command is executed, every subsequent **run** command will be profiled.

Syntax

```
profile on [-assertions] [-classes] [-cvg] [-qdas] [-solver] [-m] [-p] [-file <filename> | -fileonly <filename>]]
```

Arguments

- **-assertions**
(optional) Enables fine-grain analysis of memory capacity data collection for assertions and cover directives.
- **-classes**
(optional) Enables fine-grain analysis of memory capacity data collection for class objects.
- **-cvg**
(optional) Enables fine-grain analysis of memory capacity data collection for covergroups.
- **-qdas**
(optional) Enables fine-grain analysis of memory capacity data collection for queues, dynamic arrays, associative arrays.
- **-solver**
(optional) Enables fine-grain analysis of memory capacity data collection for randomize () calls.
- **-m**
Enables memory allocation profiling only. Optional.
- **-p**
Enables statistical performance profiling only. Optional.
- **-file <filename>**
Allows creation of a raw profile data file that can be post-processed later. Saves memory profile data into both an external file and internal data structures. Optional
- **-fileonly <filename>**
Allows creation of a raw profile data file that can be post-processed later. Saves memory profile data into an external file only, not to internal data structures. Optional

Example

- The following set of commands enables the profiler, runs the simulation for 1000 nanoseconds, and outputs the profiling data to *perf.rpt*.

```
profile on  
run 1000 ns  
profile report -file perf.rpt
```

Related Topics

- [profile clear](#)
- [profile interval](#)
- [profile off](#)
- [profile open](#)
- [profile option](#)
- [profile reload](#)
- [profile report](#)
- [profile save](#)
- [Profiling Performance and Memory Use](#)

profile open

This command opens a database containing profile data created with the [profile save](#) command. This command is useful for analyzing profile data from another user.

Syntax

```
profile open <filename>
```

Arguments

- <filename>
(required) An absolute or relative pathname to a file containing the profile database.

Related Topics

- [profile clear](#)
- [profile interval](#)
- [profile off](#)
- [profile on](#)
- [profile option](#)
- [profile reload](#)
- [profile report](#)
- [profile save](#)
- [Profiling Performance and Memory Use](#)

profile option

This command changes how profiling data are reported. The command also acts like a toggle: invoking it the first time turns on the option; invoking it a second time turns the option off. After each execution of the command the new setting is returned to the transcript.

Prerequisites

First enable profiling with the [profile on](#) command.

Syntax

```
profile option collapse_sections [on | off | status]
```

```
profile option collect_calltrees [on | off | status]
```

Arguments

- collapse_sections [on | off | status]

(required) Groups profiling data by section. A section consists of regions of code such as VHDL processes, functions, or Verilog *always* blocks. By default all profiling data are reported on a per line basis.

on — (optional) Enables profiling of data by section. (default)

off — (optional) Disables profiling of data by section

status — (optional) Returns the current setting of the **profile option collapse_sections** command.

If executed without options, the **profile option collapse_sections** command acts as a toggle.

- collect_calltrees [on | off | status]

(required) Collects data for call trees, showing which functions or routines call which others. By default this information is not collected. Simulation time and resource usage will increase if you enable collection of this data.

on — (optional) Enables profiling of data by section.

off — (optional) Disables profiling of data by section. (default)

status — (optional) Returns the current setting of the **profile option collect_calltrees** command.

Example

- Enable profiling of collapsed processes and functions.

```
profile option collapse_sections on
```

Returns:

```
# Profiling will now report collapsed processes and functions
```

- Turn off reporting of collapsed processes and functions.

profile option collapse_sections

Returns:

```
# Profiling will now NOT report collapsed processes and functions
```

Related Topics

- [profile clear](#)
- [profile interval](#)
- [profile off](#)
- [profile on](#)
- [profile open](#)
- [profile reload](#)
- [profile report](#)
- [profile save](#)
- [Profiling Performance and Memory Use](#)

profile reload

This command reads in raw profile data from an external file created during memory allocation profiling. The [profile report](#) command and the Profile and Profile Details windows of the user interface can be used to view the data. The intent of the raw profile files is to allow analysis of memory profile data in cases where the memory required for the design plus the memory required for internal profiling data exceeds the memory capacity of the machine.

The **profile reload** command clears all performance and memory profiling data collected up to that point. Any currently loaded design is terminated (similar to the **quit -sim** command), and run-time profiling is turned off. If a new design is loaded after you have read the raw profile data, then all internal profile data is cleared, but run-time profiling is not turned back on.

Prerequisites

You must first execute [profile on -m -file <filename>](#) or [profile on -m -fileonly <filename>](#).

Syntax

```
profile reload <filename>
```

Arguments

- <filename>
(required) Designates the name of the external file where raw profile data will be saved.

Related Topics

- [profile clear](#)
- [profile interval](#)
- [profile off](#)
- [profile on](#)
- [profile open](#)
- [profile option](#)
- [profile report](#)
- [profile save](#)
- [Profiling Performance and Memory Use](#)

profile report

This command outputs profiling data that have been gathered up to the point that you execute the command.

Prerequisites

First enable profiling using the [profile on](#) command or the `vsim -memprof` command.

Syntax

profile report

```
[-callercallee <func> |  
  -calltree |  
  -du [<du_name>] [-showcalls] |  
  -functoinst <func> |  
  -instofdef <inst> [-inclusiveDuMatch 0 |1] |  
  -ranked |  
  -structural [-level <positive_integer>] [<rootname>] [-showcalls]]  
[-assertions]  
  [-classes] [-cvg] [-cutoff <percentage>] [-file <filename>] [-m] [-onexit] [-p] [-qdas]  
  [-solver]
```

Arguments

- -assertions
(optional) Reports memory usage data for SystemVerilog assertions and cover directives.
- -calltree
(optional) Reports a hierarchical callstack list of statistical performance and memory allocation data. (default)
- -callercallee <func>
(optional) Creates a ranked report of all callers and callees of the specified function.
 <func> — A function name (for Systemc, PLI, FLI) or a <.v/.vhd-filename>:<line#>
- -classes
(optional) Reports memory usage data for the current number of objects allocated, the current memory allocated for class object, the peak memory allocated and peak time.
- -cutoff <percentage>
(optional) Filters out entries in the report that had less than <percentage> of time spent in them where the default is to report all entries (i.e., 0%) when -cutoff is not specified.
 <percentage> — (required) Any non-negative integer 0 - 100.
- -cvg
(optional) Reports memory usage data for the number of covergroups, cross, bins and memory allocated.

- `-du [<du_name>] [-showcalls]`
 (optional) Reports a list of statistical performance and memory allocation data organized by design unit. Optional.
 <du_name> — (optional) Reports information about a specific design unit only. If omitted, the report includes all design units.
 -showcalls — (optional) Lists function callstacks beneath each design unit. If omitted, functional callstacks are not shown in the report.
- `-file <filename>`
 (optional) Specifies that report data be saved to a file. Default is to write the report to the Transcript window.
 <filename> — Any valid filename. May include special characters and numbers.
- `-functoinst <func>`
 (optional) Creates a ranked profile report of all instances of the specified function.
 <func> — A function name (for Systemc, PLI, FLI) or a <.v/.vhd-filename>:<line#>
- `-instofdef <inst>`
 (optional) Creates a ranked report of all instances with the same definition as the specified instance, showing profile results for each.
 <inst> — The hierarchical pathname of the specified instance.
- `-inclusiveDuMatch 0 |1`
 (optional) Determines how strict the instance definition is for the **-instofdef <inst>** argument.
 0 — Includes in the report only instances that reference the exact design unit (for example, a specific entity/architecture pair).
 1 — (default) Includes all instances that reference design units with the same primary name. For example if your design has multiple architectures for a VHDL entity, a value of 1 will cause matching for all instances that use the same entity.
- `-m`
 (optional) Displays memory allocation data in the report. If **-m** is not specified, the profile report will include memory allocation data if the memory profiler was previously enabled and memory information was collected during a run.
- `-onexit`
 (optional) Causes the command to be executed when the simulator exits. Allows you to queue multiple **profile report** commands.
- `-p`
 (optional) Displays statistical performance samples in the report. If **-p** is not specified, the profile report will include performance statistics if the performance profiler was previously enabled and profile samples were collected during a run.

- **-qdas**
(optional) Reports memory usage data for queues, dynamic arrays, and associative arrays.
- **-ranked**
(optional) Reports a ranked list of statistical performance and memory allocation data.
- **-solver**
(optional) Reports memory usage data for calls to `randomize()` and memory usage.
- **-structural [-level <positive_integer>] [<rootname>] [-showcalls]**
(optional) Reports a structural list of statistical performance and memory allocation data.
 - level <positive_integer>** — (optional) Determines how far to expand instance hierarchy. If omitted, the report includes all levels.
<positive_integer> — Any positive integer.
 - <rootname>** — (optional) Causes the report to be rooted at the specified instance. If not specified, the report contains all roots and any orphan samples.
 - showcalls** — (optional) Lists function callstacks beneath each instance. If omitted, functional callstacks are not shown in the report.

Examples

- This set of commands enables the statistical sampling profiler, runs the simulation for 1000 nanoseconds, and outputs the calltree profiling data to a file named *perf.rpt*.

```
profile on  
run 1000 ns  
profile report -file perf.rpt
```

- Output ranked profile data for instances accounting for greater than 2% of the simulation time.

```
profile report -ranked -cutoff 2
```

- Output to file *perf.rpt* ranked profile data for all instances that use the same entity as does instance */top/c/s0*.

```
profile report -file perf.rpt -instofdef /top/c/s0
```

Related Topics

- [profile clear](#)
- [profile interval](#)
- [profile off](#)
- [profile on](#)
- [profile open](#)
- [profile option](#)
- [profile reload](#)
- [profile save](#)
- [Profiling Performance and Memory Use](#)

profile save

This command saves the profile data to an external database. This command is useful for sending to another user for analysis.

Use the [profile open](#) command to gain access to the database.

Prerequisites

Profiling must be active when this command is invoked. Use the [profile on](#) command to begin profiling.

Syntax

```
profile save <filename>
```

Arguments

- <filename>
(required) An absolute or relative path and filename to save the profile database to.

Related Topics

- [profile clear](#)
- [profile interval](#)
- [profile off](#)
- [profile on](#)
- [profile open](#)
- [profile option](#)
- [profile reload](#)
- [profile report](#)
- [Profiling Performance and Memory Use](#)

project

This command is used to perform common operations on projects.

Prerequisites

Some arguments to this command require a project to be opened with either the **project new** or **project open** command. Some argument must be used outside of a simulation session. Please read the argument descriptions for more information.

Syntax

```
project [addfile <filename> [<file_type>] [<folder_name>]] | [addfolder <foldername>
  [<folder_parent>]] | [calculateorder] | [close] | [compileall [-n]] | [compileorder] |
  [compileoutofdate [-n]] | [delete <filename>] | [env] | [history] | [new <home_dir>
  <proj_name> [<defaultlibrary>] [<intialini>] [0 | 1]] | [open <project>] | [removefile
  <filename>]
```

Arguments

- `addfile <filename> [<file_type>] [<folder_name>]`
(optional) Adds the specified file to the current project. Requires a project to be open.
 - `<filename>` — (required) The name of an existing file.
 - `<file_type>` — (optional) The HDL file type of the file being added. For example do for a `.do` file.
 - `<folder_name>` — (optional) Places the file in an existing folder created with **project addfolder** command. If no folder name is specified the file will be placed in the top level folder.
- `addfolder <foldername> [<folder_parent>]`
(optional) Creates a project folder within the project. Requires a project to be open.
 - `<foldername>` — (required) Any string.
 - `<folder_parent>` — (optional) Places `<foldername>` in an existing parent folder. If `<folder_parent>` is unspecified, `<foldername>` is placed at the top level.
- `calculateorder`
(optional) Determines the compile order for the project by compiling each file, then moving any compiles that fail to the end of the list. This is repeated until there are no more compile errors.
- `close`
(optional) Closes the current project.
- `compileall [-n]`
(optional) Compiles all files in the project using the defined compile order.
 - `-n` — (optional) Returns a list of the compile commands this command would execute, without actually executing the compiles.

project

- **compileorder**
(optional) Returns the current compile order list.
- **compileoutofdate [-n]**
(optional) Compiles all files that have a newer date/time stamp than the last time the file was compiled.
 - n — Returns a list of the compile commands this command would execute, without actually executing the compiles.
- **delete <filename>**
(optional) Deletes a project file.
 - <filename> — Any *.mpf* file.
- **env**
(optional) Returns the current project file and path.
- **history**
(optional) Lists a history of manipulated projects. Must be used outside of a simulation session.
- **new <home_dir> <proj_name> [<defaultlibrary>] [<initialini>] [0 | 1]**
(optional) Creates a new project under a specified home directory with a specified name and optionally a default library. The name of the work library will default to "work" unless specified. A new project cannot be created while a project is currently open or a simulation is in progress.
 - <home_dir> — The path to the new project directory within the current working directory.
 - <proj_name> — Specifies a name for the new project. The file will be saved as an *.mpf* file
 - <defaultlibrary> — Specifies a name for the default library.
 - <initialini> — An optional *modelsim.ini* file can be specified as a seed for the project file by using the initialini option. If initialini is an empty string, then ModelSim uses the current *modelsim.ini* file when creating the project. You must specify a default library if you want to specify initialini.
 - 0 — (default) Copies all library mappings from the specified <initialini> file into the new project.
 - 1 — Copies library mappings referenced in an "others" clause in the initial *.ini* file.
- **open <project>**
(optional) Closes any currently opened project and opens a specified project file (must be a valid *.mpf* file), making it the current project. Changes the current working directory to the project's directory. Must be used outside of a simulation session.

- `removefile <filename>`
(optional) Removes the specified file from the current project.

Examples

- Make `/user/george/design/test3/test3.mpf` the current project and changes the current working directory to `/user/george/design/test3`.

```
project open /user/george/design/test3/test3.mpf
```

- Execute current project library build scripts.

```
project compileall
```

property list

This command changes one or more properties of the specified signal, net, or register in the List window.

The properties correspond to those you can set by selecting **View > Signal Properties** (List window). At least one argument must be used.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
property list [-window <wname>] [-label <label>] [-radix <type>]  
             [-radixenumnumeric | -radixenumsymbolic] [-trigger 0 | 1] [-width <number>] <pattern>
```

Arguments

- -window <wname>
(optional) Specifies a particular List window when multiple instances of the window exist (e.g., list2). If no window is specified the default window is used; the default window is determined by the most recent invocation of the [view](#) command.

<wname> — Specifies a List window other than the default list window.

- -label <label>
(optional) Specifies a label to appear at the top of the List window column identifying <pattern>.

<label> — A user-specified name. Can include numbers and special characters.

- -radix <type>
(optional) Specifies the radix for List window objects.
<type> — Any valid radix type: binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default. If no radix is specified for an enumerated type, the default representation is used.

You can change the default radix for the current simulation using the [radix](#) command. You can change the default radix permanently by editing the [DefaultRadix](#) variable in the *modelsim.ini* file.

If you specify a radix for an array of a VHDL enumerated type, ModelSim converts each signal value to 1, 0, Z, or X.

- -radixenumnumeric
(optional) Causes Verilog and SystemC enums to be displayed as numbers (formatted by the current radix). This overrides the default behavior of always showing enums symbolically.
- -radixenumsymbolic
(optional) Restores the default behavior of displaying Verilog and SystemC enums as symbols by reversing the action of the -radixenumnumeric option.

-
- **-trigger 0 | 1**
(optional) Toggles whether changes to <pattern> add data in the list window.
 - 0 — Does not write changes to <pattern> to the list window.
 - 1 — Writes changes to <pattern> to the list window.
 - **-width <number>**
(optional) Specifies the desired column width for the objects matching the specified pattern.
 - <number> — Any positive integer 1 through 256.
 - **<pattern>**
(required) Specifies a name or wildcard pattern to match the full pathnames of the signals, nets, or registers for which you are defining the property change. Must be specified as the final argument to the **property list** command.

property wave

This command changes one or more properties of the specified signal, net, or register in the Wave window.

The properties correspond to those you can set by selecting **View > Signal Properties** (Wave window). At least one argument must be used.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
property wave [-window <wname>] [-color <color>] [-format {analog | literal | logic}]  
  [-height <number>] [-offset <number>] [-radix <type>] [S | -radixenumsymbolic]  
  [-scale <n>] <pattern>
```

Arguments

- -window <wname>
(optional) Specifies a particular Wave window when multiple instances of the window exist. If no window is specified the default window is used; the default window is determined by the most recent invocation of the [view](#) command.

<wname> — Specifies the instance of the Wave window to use (for example, wave2).
- -color <color>
(optional) Specifies a different color is to be used for the waveform.

<color> — The color value may be a color name or its hex value.
- -format {analog | literal | logic}
(optional) Specifies the format of the wave form in <pattern>.

analog — (optional) Displays a waveform whose height and position is determined by the **-scale** and **-offset** arguments.

literal — (optional) Displays the waveform as a box containing the object value (if the value fits the space available).

logic — (optional) Displays values as 0, 1, X, or Z.
- -height <number>
(optional) Specifies the height (in pixels) of the waveform.

<number> — Any integer.
- -offset <number>
(optional) Specifies the waveform position offset in pixels. Valid only when **-format** is specified as **analog**.

<number> — Any non-negative integer.

- `-radix <type>`

(optional) Specifies the radix for Wave window objects.

`<type>` — Any valid radix type (or unique abbreviation): binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default. If no radix is specified for an enumerated type, the default representation is used.

You can change the default radix for the current simulation using the [radix](#) command. You can change the default radix permanently by editing the [DefaultRadix](#) variable in the *modelsim.ini* file.

If you specify a radix for an array of a VHDL enumerated type, ModelSim converts each signal value to 1, 0, Z, or X.

- `-radixenumnumeric`

(optional) Causes Verilog and SystemC enums to be displayed as numbers (formatted by the current radix). This overrides the default behavior of always showing enums symbolically.

- `-radixenumsymbolic`

(optional) Restores the default behavior of displaying Verilog and SystemC enums as symbols by reversing the action of the `-radixenumnumeric` option.

- `-scale <n>`

(optional) Specifies the waveform scale relative to the unscaled size value of 1. Valid only when **-format** is specified as **analog**.

`<n>` — Any non-negative integer.

- `<pattern>`

(required) Specifies a name or wildcard pattern to match the full path names of the signals, nets, or registers for which you are defining the property change. Must be specified as the final argument to the **property wave** command

push

This command moves the specified number of call frames down the C callstack.

This command is used with C Debug. Refer to “[C Debug](#)” for more information.

Syntax

```
push <#_of_levels>
```

Arguments

- `<#_of_levels>`
(optional) Specifies the number of call frames to move down the C callstack. If unspecified, 1 level is assumed.

Examples

- Move down 1 call frame.

```
push
```

- Move down 4 call frames.

```
push 4
```

Related Topics

- [pop](#)
- [C Debug](#)

pwd

This Tcl command displays the current directory path in the Transcript window.

Syntax

```
pwd
```

Arguments

- None

quietly

This command turns off transcript echoing for the specified command.

Syntax

quietly <command>

Arguments

- <command>
(required) Specifies the command for which to disable transcript echoing. Any results normally echoed by the specified command will not be written to the Transcript window. To disable echoing for all commands use the [transcript](#) command with the **-quietly** option.

Related Topics

- [transcript](#)

quit

This command exits the simulator.

If you want to stop the simulation using a [when](#) command, you must use a [stop](#) command within your when statement, you must not use an [exit](#) or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated.

Syntax

```
quit [-f | -force] [-sim] [-code <integer>]
```

Arguments

- -f | -force
(optional) Quits without asking for confirmation. If omitted, ModelSim asks you for confirmation before exiting. (The -f and -force arguments are equivalent.)
- -sim
(optional) Unloads the current design in the simulator without exiting ModelSim. All files opened by the simulation will be closed including the WLF file (*vsim.wlf*).
- -code <integer>
(optional) Quits the simulation and issues an exit code.

 <integer> — This is the value of the exit code. You should not specify an exit code that already exists in ModelSim. Refer to the section "[Exit Codes](#)" in the User's Manual for a list of existing exit codes. You can also specify a variable in place of <integer>. You should always print a message before running the quit -code command to explicitly state the reason for exiting.

Examples

Refer to the Examples section of the [exit](#) command for an example of using the **-code** argument. The quit and exit commands behave similarly in this regard.

qverilog

The **qverilog** command compiles (**vlog**), optimizes (**vopt**), and simulates (**vsim**) Verilog and SystemVerilog designs in a single step. It combines the compile, elaborate, and simulate phases together, as some users may be accustomed to doing with NC-Sim. This command is provided to ease these users' transition to ModelSim.

The **qverilog** command invokes **vlog**, **vopt**, and then **vsim**. All standard **vlog** (and **vopt**) arguments are supported and are applied directly to the **qverilog** command line. All **vsim** options are supported and are applied through the **qverilog -R** argument.

You can directly enter either C or C++ file onto the **qverilog** command line. ModelSim automatically processes them using the SystemVerilog Direct Programming Interface (DPI). Refer to “[DPI and the qverilog Command](#)” for details. If your design contains DPI export tasks or functions, it is recommended that you use the vlog/vsim flow.

You can invoke the GUI by specifying the **-gui** argument through the **qverilog -R** argument.

By default, **qverilog** runs the simulation and quits automatically by invoking an implicit "run -all; quit -f". However, if you invoke **qverilog** with **-do**, **-gui**, or **-i**, **qverilog** invokes the simulator and keeps it open until you explicitly quit ModelSim.

The **qverilog** command creates a work library named **work** in the current directory, if not present already.

The command arguments listed below are only those unique to the **qverilog** command. This command also supports all **vlog** command arguments.

Syntax

```
qverilog [[<vlog_and_vopt_options>]] [-ccflags "opts"] [-gui] [-l <logfile>] <filename>  
      [-ldflags "opts"] [-R <vsim_options>] [-work <library_name>]
```

Arguments

- [**<vlog_and_vopt_options>**]

All **vlog** and **vopt** options are supported. For example, if you are running qverilog on a SystemVerilog design, you need to add the **-sv** argument to the command line.

- **-ccflags "opts"**

Specifies all C/C++ compiler options. Options are in quotes. Optional.

For **-ccflags** and **-ldflags**, **qverilog** does not check the validity of the option(s) you specify. The options are directly passed on to the compiler and linker, and if they are not valid, an error message is generated by the compiler/linker.

- **-gui**

Simulates the design using the ModelSim GUI.

- **-l <logfile>**

Creates a logfile/transcript compatible with vlog -l logfile. Optional. If omitted, a default transcript called *qverilog.log* is created that collects the output from **vlog**, **vopt**, and **vsim**.

- `<filename>`
Specifies the name of the Verilog or C/C++ source code file to compile. One filename is required. Multiple filenames separated by spaces may be entered. Wildcards may be used. In the case of C files, they are automatically processed as DPI code.
- `-ldflags "opts"`
Specifies all linker options in quotes. Optional.
For **-cflags** and **-ldflags**, **qverilog** does not check the validity of the option(s) you specify. The options are directly passed on to the compiler and linker, and if they are not valid, an error message is generated by the compiler/linker.
- `-R <vsim_options>`
Specifies valid **vsim** arguments to be applied to the simulation. All `vlog` and `vopt` arguments must come before `-R` is specified, as all arguments specified after `-R` are interpreted as `vsim` arguments.
- `-work <library_name>`
Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional. By default, the compiled design units are added to the **work** library. The specified pathname overrides the pathname specified for `work` in the project file.

Examples

- Compile, optimize, and simulate the specified files. The C/C++ code contained in the `d.c` file is processed as DPI code, creating a shared object, and loading it into `vsim` at runtime. Creates a logfile named "logfile" and opens the ModelSim GUI with the simulation loaded and ready to run.

```
qverilog -l logfile a.v b.v c.v d.c -R -gui
```

radix

This command specifies the default radix to be used for the current simulation. Specifying the command with no argument returns the current radix setting.

The command can be used at any time. The specified radix is used for all commands ([force](#), [examine](#), [change](#), etc.) as well as for displayed values in the Objects, Locals, Dataflow, Schematic, List, and Wave windows.

Alternate methods for changing the default radix:

- In the *modelsim.ini* file, edit the [DefaultRadix](#) variable.
- Choose **Simulate > Runtime Options** from the main menu, click the **Defaults** tab, make your selection in the **Default Radix** box.

Syntax

```
radix [-symbolic | -binary | -octal | -decimal | -hexadecimal | -unsigned | -ascii | -time]
      [-enumnumeric | -enumsymbolic]
```

Arguments

You can abbreviate the following arguments to any length. For example, -dec is equivalent to -decimal.

- -symbolic
(optional) Displays values in a form closest to their natural form.
- -binary
(optional) Displays values in binary format.
- -octal
(optional) Displays values in octal format.
- -decimal
(optional) Displays values in decimal format. You can specify -signed as an alias for this argument.
- -hexadecimal
(optional) Displays values in hexadecimal format.
- -unsigned
(optional) Displays values in unsigned decimal format.
- -ascii
(optional) Display a Verilog object as a string equivalent using 8-bit character encoding.
- -time
(optional) Displays values of time for register-based types in Verilog.

- **-enumnumeric**
(optional) Causes Verilog and SystemC enums to be displayed as numbers (formatted by the current radix). This overrides the default behavior of always showing enums symbolically.
- **-enumsymbolic**
(optional) Restores the default behavior of displaying Verilog and SystemC enums as symbols by reversing the action of the **-enumnumeric** option.

Related Topics

- [User-Defined Radices](#)
- [radix define](#)
- [radix delete](#)
- [radix names](#)
- [radix list](#)
- [radix signal](#)

radix define

This command is used to create or modify a user-defined radix. A user definable radix is used to map bit patterns to a set of enumeration labels. User-defined radices are available for use in the Wave and List windows or with the [examine](#) command.

Syntax

```
radix define <name> <definition_body> [-color <value>]
```

Arguments

- <name>
(required) User-specified name for the radix.
- <definition_body>
(required) A list of number pattern, label pairs. The definition body has the form:

```
{  
    <numeric-value> <enum-label>,  
    <numeric-value> <enum-label>  
    -default <radix_type>  
}
```

A <numeric-value> is any legitimate HDL integer numeric literal. To be more specific:

```
<base>#<base-integer># --- <base> is 2, 8, 10, or 16  
<base>"bit-value" --- <base> is B, 0, or X  
<integer>  
<size>'<base><number> --- <size> is an integer, <base> is b, d, o,  
or h.
```

Refer to the Verilog and VHDL LRMs for exact definitions of these numeric literals.

The comma (,) in the definition body is optional. The <enum-label> is any arbitrary string. It should be surrounded by quotation marks (""), especially if it contains spaces.

The **-default** entry is optional. If present, it defines the radix to use if a match is not found for a given value. The **-default** entry can appear anywhere in the list, it does not have to be at the end.

- -color <value>
(optional) Designates a color for the waveform and text in the Wave window.
<value> — The color value may be a color name or its hex value (see example below).

Example

- The radix define command used to create a radix called “States,” which will display state values in the List, Watch, and Wave windows instead of numeric values.

```
radix define States {  
    11'b00000000001 "IDLE",  
    11'b00000000010 "CTRL",
```

```

11'b00000000100 "WT_WD_1",
11'b000000001000 "WT_WD_2",
11'b000000010000 "WT_BLK_1",
11'b000000100000 "WT_BLK_2",
11'b00001000000 "WT_BLK_3",
11'b00010000000 "WT_BLK_4",
11'b00100000000 "WT_BLK_5",
11'b01000000000 "RD_WD_1",
11'b10000000000 "RD_WD_2",
-default hex
}

```

- The following example illustrates how to specify the radix color:

```

radix define States {
11'b00000000001 "IDLE" -color yellow,
11'b00000000010 "CTRL" -color #ffee00,
11'b00000000100 "WT_WD_1" -color orange,
11'b000000001000 "WT_WD_2" -color orange,
11'b000000010000 "WT_BLK_1",
11'b000000100000 "WT_BLK_2",
11'b00001000000 "WT_BLK_3",
11'b00010000000 "WT_BLK_4",
11'b00100000000 "WT_BLK_5",
11'b01000000000 "RD_WD_1" -color green,
11'b10000000000 "RD_WD_2" -color green,
-default hex
-defaultcolor white
}

```

If a pattern/label pair does not specify a color, the normal wave window colors will be used. If the value of the waveform does not match any pattern, then the **-default radix** and **-defaultcolor** will be used.

To specify a range of values, wildcards may be specified for bits or characters of the value. The wildcard character is "?", similar to the iteration character in a Verilog UDP, for example:

```
radix define {  
    6'b01??00 "Write" -color orange,  
    6'b10??00 "Read" -color green  
}
```

In this example, the first pattern will match "010000", "010100", "011000", and "011100". In case of overlaps, the first matching pattern is used, going from top to bottom.

Related Topics

- [User-Defined Radices](#)
- [radix](#)
- [radix delete](#)
- [radix names](#)
- [radix list](#)
- [radix signal](#)

radix delete

This command will remove the radix definition from the named radix.

Syntax

```
radix delete <name>
```

Arguments

- <name>
(required) Removes the radix definition from the named radix.

Related Topics

- [User-Defined Radices](#)
- [radix](#)
- [radix define](#)
- [radix list](#)
- [radix names](#)
- [radix signal](#)

radix list

This command will return the complete definition of a radix, if a name is given. If no name is given, it will list all the defined radices.

Syntax

```
radix list [<name>]
```

Arguments

- <name>
(optional) Returns the complete definition of the named radix.

Related Topics

- [User-Defined Radices](#)
- [radix](#)
- [radix define](#)
- [radix delete](#)
- [radix names](#)
- [radix signal](#)

radix names

This command returns a list of currently defined radix names.

Syntax

radix names

Arguments

None

Related Topics

- [User-Defined Radices](#)
- [radix](#)
- [radix define](#)
- [radix delete](#)
- [radix list](#)
- [radix signal](#)

radix signal

This command sets or inspects radix values for the specified signal in the Objects, Locals, and Wave windows.

Note



The intent is for this command to be used for a small number of signals. If the majority of signals in a design are to use a particular radix value, then set that value as the default radix with the [radix](#) command, and use the **radix signal** command for the rest.

When no argument is used, the **radix signal** command returns a list of all signals with a radix.

Syntax

```
radix signal [<signal_name> [<radix_value>]]
```

Arguments

- <signal_name>
(optional) Name of the signal for which the radix will be set (if <radix_value> is specified) or inspected.
- <radix_value>
(optional) Value of the radix to be set for the specified signal. Use empty quotation marks ("") to unset the radix for the specified signal.

Related Topics

- [User-Defined Radices](#)
- [radix](#)
- [radix define](#)
- [radix list](#)
- [radix delete](#)

readers

This command displays the names of all readers of the specified object.

The reader list is expressed relative to the top-most design signal/net connected to the specified object.

Syntax

```
readers <object_name>
```

Arguments

- <object_name>
(required) Specifies the name of the signal or net whose readers are to be shown. All signal or net types are valid. Multiple names and wildcards are accepted.

Related Topics

- [drivers](#)

report

This command displays information relevant to the current simulation.

Syntax

report files

report where [ini] [pwd] [transcript] [wlf] [project]

report simulator control

report simulator state

Arguments

- files
Returns a list of all source files used in the loaded design. This information is also available in the Specified Path column of the Files window.
- where [ini] [pwd] [transcript] [wlf] [project]
Returns a list of configuration files where the arguments limit the list to those files specified. If specified without arguments, returns a list of all configuration files in the current simulation.
 - ini — (optional) Returns the location of the *modesim.ini* file.
 - pwd — (optional) Returns the current working directory.
 - transcript — (optional) Returns the location for saving the transcript file.
 - wlf — (optional) Returns the current location for saving the *.wlf* file.
 - project — (optional) Returns the current location of the project file.
- simulator control
Displays the current values for all simulator control variables.
- simulator state
Displays the simulator state variables relevant to the current simulation.

Examples

- Display configuration file information

report where

Returns:

```
# INI {modelsim.ini}
# PWD ./Testcases/
# Transcript transcript
# WLF vsim.wlf
# Project {}
```

- Display all simulator control variables.

report simulator control

Returns:

```
# UserTimeUnit = ns
# RunLength =
# IterationLimit = 5000
# BreakOnAssertion = 3
# DefaultForceKind = default
# IgnoreNote = 0
# IgnoreWarning = 0
# IgnoreError = 0
# IgnoreFailure = 0
# IgnoreSVAInfo= 0
# IgnoreSVAWarning = 0
# IgnoreSVAError = 0
# IgnoreSVAFatal = 0
# CheckpointCompressMode = 1
# NumericStdNoWarnings = 0
# StdArithNoWarnings = 0
# PathSeparator = /
# DefaultRadix = symbolic
# DelayFileOpen = 1
# WLFfilename = vsim.wlf
# WLFTimeLimit = 0
# WLFSizeLimit = 0
```

You can set these simulator control variables to a new value by using the Tcl [set Command Syntax](#).

- Display all simulator state variables. Only the variables that relate to the design being simulated are displayed:

report simulator state

Returns:

```
# now = 0.0
# delta = 0
# library = work
# entity = type_clocks
# architecture = full
# resolution = 1ns
```

Viewing preference variables

Preference variables have more to do with the way things look (but not entirely) rather than controlling the simulator. You can view preference variables from the Preferences dialog box. Select **Tools > Edit Preferences** (Main window).

Related Topics

- [modelsim.ini Variables](#)
- [Simulator GUI Preferences](#)

restart

This command reloads the design elements and resets the simulation time to zero. Only design elements that have changed are reloaded. (Note that SDF files are always reread during a restart.)

- If no design is loaded, the **restart** command produces a message to that effect and takes no further action.
- If a simulation is loaded, the **restart** command restarts the simulation.
- If multiple datasets are open, including a simulation, the environment is changed to the simulation context and the simulation is restarted.

Shared libraries are handled as follows during a restart:

- Shared libraries that implement VHDL foreign architectures only are reloaded at each restart when the architecture is elaborated (unless **vsim -keeploaded** is used).
- Shared libraries loaded from the command line (**-foreign** and **-pli** options) and from the Veriuser entry in the *modelsim.ini* file are reloaded (unless **vsim -keeploaded** is used).
- Shared libraries that implement VHDL foreign subprograms remain loaded (they are not reloaded) even if they also contain code for a foreign architecture.

You can configure defaults for the restart command by setting the **DefaultRestartOptions** variable in the *modelsim.ini* file. Refer to “[Restart Command Defaults](#)”.

To handle restarts with Verilog PLI applications, you need to define a Verilog user-defined task or function, and register a misctf class of callback. To handle restarts with Verilog VPI applications, you need to register reset callbacks. To handle restarts with VHDL FLI applications, you need to register restart callbacks. Refer to “[Verilog Interfaces to C](#)” for more information on the Verilog PLI/VPI/DPI and the *ModelSim FLI Reference* for more information on the FLI.

Syntax

```
restart [-force] [-nobreakpoint] [-nolist] [-nolog] [-nowave] [-noassertions]
```

Arguments

- **-force**
(optional) Specifies that the simulation will be restarted without requiring confirmation in a popup window.
- **-noassertions**
(optional) Specifies that current assert directive configurations will not be maintained after the simulation is restarted where the default is for assert directive settings to be maintained after the simulation is restarted.

- **-nobreakpoint**
(optional) Specifies that all breakpoints will be removed when the simulation is restarted where the default is for all breakpoints to be reinstalled after the simulation is restarted.
- **-nolist**
(optional) Specifies that the current List window environment will **not** be maintained after the simulation is restarted where the default is for all currently listed HDL objects and their formats to be maintained.
- **-nolog**
(optional) Specifies that the current logging environment will **not** be maintained after the simulation is restarted where the default is for all currently logged objects to continue to be logged.
- **-nowave**
(optional) Specifies that the current Wave window environment will **not** be maintained after the simulation is restarted where the default is for all objects displayed in the Wave window to remain in the window with the same format.

Related Topics

- [checkpoint](#)
- [restore](#)
- [vsim](#)
- [Checkpointing and Restoring Simulations](#)

restore

This command restores the state of a simulation that was saved with a checkpoint command during the current invocation of **vsim** (called a "warm restore").

The items restored are: simulation kernel state, *vsim.wlf* file, HDL objects listed in the List and Wave windows, file pointer positions for files opened under VHDL and under Verilog \$fopen, and the saved state of foreign architectures.

If you want to restore while running **vsim**, use this command. If you want to start up **vsim** and restore a previously-saved checkpoint, use **vsim -restore** (called a "cold restore").

Checkpoint/restore allows a cold restore, followed by simulation activity, followed by a warm restore back to the original cold-restore checkpoint file. Warm restores to checkpoint files that were not created in the current run are not allowed except for this special case of an original cold restore file.

Checkpoint files are platform dependent—you cannot checkpoint on one platform and restore on another.

Syntax

```
restore <filename>
```

Arguments

- <filename>
(required) Specifies the name of the checkpoint file.

Related Topics

- [checkpoint](#)
- [vsim](#)
- [Checkpointing and Restoring Simulations](#)

resume

This command is used to resume execution of a macro (DO) file after a [pause](#) command or a breakpoint.

This command may be input manually or placed in an [onbreak](#) command string. (Placing a **resume** command in a [bp](#) command string does not have this effect.) The **resume** command can also be used in an [onerror](#) command string to allow an error message to be printed without halting the execution of the macro file.

Syntax

resume

Arguments

- None

Related Topics

- [abort](#)
- [do](#)
- [onbreak](#)
- [onerror](#)
- [pause](#)
- [Useful Commands for Handling Breakpoints and Errors](#)
-

right

This command searches right (next) for signal transitions or values in the specified Wave window.

It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a waveform takes on a particular value, or an expression of multiple signals evaluates to true. See the [left](#) command for related functionality.

The procedure for using **right** entails three steps:

1. Click on the desired waveform.
2. Click on the desired starting location.
3. Issue the **right** command. (The [seetime](#) command can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

Note



“[Wave Window Mouse and Keyboard Shortcuts](#)” are also available for next and previous edge searches. Tab searches right (next) and shift-tab searches left (previous).

Syntax

```
right [-expr {<expression>}] [-falling] [-noglitch] [-rising] [-value <sig_value>]  
      [-window <wname>] [<n>]
```

Arguments

- -expr {<expression>}
(optional) Searches the waveform display for an expression. When the search evaluates to a boolean true, the active cursor moves to the found location. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced Wave window. A signal may be specified either by its full path or by the shortcut label displayed in the Wave window.
{<expression>} — Any valid expression. See [GUI_expression_format](#) for the format of the expression. The expression must be placed within curly braces.
- -falling
(optional) Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored.
- -noglitch
(optional) Looks at signal values only on the last delta of a time step. For use with the **-value** option only.

- **-rising**
(optional) Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored.
- **-value <sig_value>**
(optional) Specifies a value of the signal to match. Only one signal may be selected, but that signal may be an array.

<sig_value> — Must be specified in the same radix that the selected waveform is displayed. Case is ignored, but otherwise the value must be an exact string match -- don't-care bits are not yet implemented.
- **-window <wname>**
(optional) Specifies an instance of the Wave window that is not the default. Otherwise, the default Wave window is used. Use the [view](#) command to change the default window.

<wname> — The name of a Wave window other than the current default window.
- **<n>**
(optional) Specifies to find the nth match. If less than n are found, the number found is returned with a warning message, and the cursor is positioned at the last match. The default is 1.

Examples

- Find the second time to the right at which the selected vector transitions to FF23, ignoring glitches.

```
right -noglitch -value FF23 2
```

- Go to the next transition on the selected signal.

```
right
```

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the [GUI_expression_format](#).

- Search right for an expression that evaluates to a boolean 1 when signal *clk* just changed from low to high and signal *mystate* is the enumeration reading and signal */top/u3/addr* is equal to the specified 32-bit hex constant; otherwise is 0.

```
right -expr {clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)}
```

- Search right for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal */top/u3/addr* equals hex ac.

```
right -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
```

- Search right for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, and clock just changed from low to high and signal *mode* is enumeration writing.

```
right -expr {{{(NOW > 23 us) && (NOW < 54 us)}} && clk'rising && (mode == writing)}
```

Related Topics

- [GUI_expression_format](#)
- [left](#)
- [seetime](#)
- [view](#)

run

This command advances the simulation by the specified number of timesteps.

Syntax

```
run [<timesteps>[<time_units>] | -all | -continue | -finish | -init | -next | -over [<n>] |  
-step [-inst <full_path>] [-out] [-over [<n>]] [-this "this==<class_handle>"]]
```

Arguments

- No arguments
Runs the simulation for the default time (100 ns).
You can change the default <timesteps> and <time_units> in the GUI with the Run Length toolbar box in the Simulate toolbar or from the *modelsim.ini* file: [RunLength](#) and [UserTimeUnit](#) variables.
- <timesteps>[<time_units>]
(optional) Specifies the number of timesteps for the simulation to run. The number may be fractional, or may be specified as absolute by preceding the value with the character @.

 <time_units> — Any valid time unit: fs, ps, ns, us, ms, or sec where the default is to use the current time unit.
- -all
(optional) Causes the simulator to run the current simulation forever, or until it hits a breakpoint or specified break event.
- -continue
(optional) Continues the last simulation run after a run -step, run -step -over command or a breakpoint. A **run -continue** command may be input manually or used as the last command in a [bp](#) command string.
- -final
(optional) Instructs the simulator to run all final blocks then exit the simulation.
- -finish
(optional) In [C Debug](#) only, continues the simulation run and returns control to the calling function.
- -init
(optional) Initializes non-trivial static SystemVerilog variables before beginning the simulation, for example, expressions involving other variables and function calls,. This could be useful for when you want to initialize values before executing any [force](#), [examine](#), or [bp](#) commands.

You cannot use **run -init** after any other run commands or if you have specified [vsim -runinit](#) on the command line because all variables would have been initialized by that point.

run

- -next

(optional) Causes the simulator to run to the next event time.

- -step

(optional) Steps the simulator to the next HDL or C statement. Current values of local HDL variables may be observed at this time using the Locals window. You can specify the following arguments when you use -step:

-inst <full_path>

(optional) Instructs the simulation to step into a specific instance, process, or thread.

<full_path> — Specifies the full path to an instance, process or thread.

-out

(optional) Instructs the simulation to step out of the current function or procedure and return to the caller.

-over [<n>]

(optional) Directs ModelSim to run VHDL procedures and functions, Verilog tasks and functions, and C functions but to treat them as simple statements instead of entering and tracing them line by line.

<n> — (optional) Moves the debugger <n> lines ahead when you are using C Debug. specified as a positive integer value.

You can use the **-over** argument to skip over a VHDL procedure or functions, Verilog task or functions, or a C function. When a wait statement or end of process is encountered, time advances to the next scheduled activity. ModelSim then updates the Process and Source windows to reflect the next activity.

-this "this==<class_handle>"

(optional) Instructs the simulation to step into a System Verilog class. To obtain the handle of the class, use the [examine](#) -handle command.

<class_handle> — Specifies a System Verilog class. Note that you must use quotation marks (“ ”) with this argument.

- -over [<n>]

(optional) Directs ModelSim to run VHDL procedures and functions, Verilog tasks and functions, and C functions but to treat them as simple statements instead of entering and tracing them line by line.

<n> — (optional) Moves the debugger <n> lines ahead when you are using C Debug. specified as a positive integer value.

Examples

- Advance the simulator 1000 timesteps.

```
run 1000
```


- Advance the simulator the appropriate number of timesteps corresponding to 10.4 milliseconds.

```
run 10.4 ms
```

- Advance the simulator to timestep 8000.

```
run @8000
```

- Advance the simulator into the instance /top/p.

```
run -step -inst /top/p
```

Related Topics

- [Simulate Toolbar](#)
- [C Debug](#)

runStatus

This command returns the current state of your simulation to stdout after issuing a `run` or `step` command.

Syntax

```
runStatus [-full]
```

Arguments

- `-full`
(optional) Appends additional information to the output of the **runStatus** command.

Returns

[Table 2-4](#) (runStatus results) and [Table 2-5](#) (runStatus -full results) show outputs of the runStatus command.

Table 2-4. runStatus Command States

State	Description
ready	The design is loaded and is ready to run.
break	The simulation stopped before completing the requested run.
error	The simulation stopped due to an error condition.
loading	The simulation is currently elaborating.
nodesign	There is no design loaded.
checkpoint	A checkpoint is being created, do not interrupt this process.
cready	The design is loaded and is ready to run in C debug mode.
initializing	The user interface initialization is in progress.

Table 2-5. runStatus -full Command Information

-full Information	Description
bkpt	stopped at breakpoint
bkpt_builtin	stopped at breakpoint on builtin process
end	reached end of requested run
fatal_error	encountered fatal error (such as, divide by 0)
iteration_limit	iteration limit reached, possible feedback loop
silent_halt	mti_BreakSilent() called,
step	run -step completed
step_builtin	run -step completed on builtin process

Table 2-5. runStatus -full Command Information

-full Information	Description
step_wait_suspend	run -step completed, time advanced.
user_break	run interrupted do to break-key or ^C (SIGINT)
user_halt	mti_Break() called.
user_stop	stop or finish requested from vpi, stop command, etc.
gate_oscillation	Verilog gate iteration limit reached.
simulation_stop	pli stop_simulation() called.

sccom

The sccom command actually provides two different functions: sccom uses an external C/C++ compiler to compile SystemC source code into the work library, while sccom -link takes compiled source code and links the design.

Compile syntax

```
sccom [-32 | -64] [-93] [<CPP compiler options>] [<CPP linker options>]
      [-cpppath <filename>] [-dumpscvext <filename>] [-dpilib <libname>]
      [-error <msg_number> [,<msg_number>,...]] [-f <filename>]
      [-fatal <msg_number> [,<msg_number>,...]] [-help] [-incr] [-lib <library_pathname>]
      [-libshared <library_pathname>] [-link] [-linkshared] [-log <logfile>]
      [-modelsimini <ini_filepath>] [-nodbgSYM] [-nodebug] [-nologo]
      [-note <msg_number> [,<msg_number>,...]] [-scms] [-scv] [-scversion]
      [-suppress <msg_number> [,<msg_number>,...]] [-vv] [-verbose] [-version]
      [-warning <msg_number> [,<msg_number>,...]] [-x c | c++] <filename>
```

Link syntax

```
sccom -link
      [<CPP linker options>] [-dpilib <libname>] [-f <filename>] [-help] [-lib <library_pathname>]
      [-log <logfile>] [-nologo] [-scv] [-vv] [-verbose] [-version] [-work <library_name>]
```

Description

You can run this command from within ModelSim, from the operating system command prompt, or during simulation.

To enable source debugging of SystemC code, you must compile for debugging by specifying the **-g** argument of the CPP compiler.

Compiled libraries have the following dependencies:

- Platform — If you move between platforms, you need to run **vdcl -allsystemc** on the working library and then recompile your SystemC source.
- Version — If you install a new release, you need to re-compile your library with the current version of **sccom**. For example, you cannot use a library compiled with v6.5 in a simulation using v6.5a **vsim**. You would have to run sccom in v6.5a to re-compile your library (**sccom -version** displays the version number of the compiler).

During the linking of the design (with **sccom -link**), the order in which you specify archives (*.a*) and object files is very important. You must specify any dependent *.a* or *.o* before the *.a* or *.o* on which it depends.

The sccom command can recognize the file type as either C or C++ by the filename extension and will use the appropriate compiler, as follows:

- gcc compiler on source files with C source extensions: *.c*, *.i*
- g++ compiler on source files with C++ extensions: *.CPP*, *.cpp*, *.C*, *.c++*, *.cc*, *.cp*, *.cxx*, *.ii*

For best performance, it is recommended to run **sccom** in multi-file compilation mode, which requires that you can write to the current working directory. By default, **sccom** works in multi-file compilation mode, passing all source files to the GNU compilers and debug generator in a single step. If the working directory has read-only permissions, **sccom** automatically performs single-file compilation, which decreases performance because only one source file is compiled at a time.

Arguments

- **-32** | **-64**

Specifies whether **sccom** uses the 32- or 64-bit executable, where **-32** is the default.

These options override the `MTI_VCO_MODE` environment variable, which applies only to executables used from the `<install_dir>/bin/` directory. Therefore, these options are ignored if you run **sccom** from an `<install_dir>/<platform>/` directory.

You can only specify these options on the command line, therefore they are not recognized as part of a file used with the `-f` switch.

- **-93**

Makes the design unit strictly case-sensitive, enforcing case-sensitivity across the SystemC-HDL mixed language boundary. Optional.

- `<C++ compiler options>`

Any normal C++ compiler option can be used, with the exception of the **-o** and **-c** options. You must specify the **-g** argument to compile for debugging. By default, **sccom** compiles without debugging information. You can specify arguments for all **sccom** compiles by editing the **CppOptions** variable in the `modelsim.ini` file.

- **-DSC_**

Specifies SystemVerilog libraries for SystemC DPI (Direct Programming Interface). Optional. Only one library can be specified per each `-dpilib` argument. See section [SystemC Procedural Interface to SystemVerilog](#).

- **-DSC_INCLUDE_MTI_AC**

Enable native debug support of Algorithmic-C datatypes. Optional.

- **-DSC_INCLUDE_DYNAMIC_PROCESSES**

Enable dynamic processes. Optional.

- **-DSC_INCLUDE_FX**

Enable fixed-point datatypes. Optional.

- **-DSC_USE_STD_STRING**

- Replace `sc_string` with `std::string`. Optional.
- `-DSC_USE_STD_STRING_OLD`
Use deprecated `sc_string`. Optional.
- `-DMTI_BIND_SC_MEMBER_FUNCTION`
Enable registration of module member functions as DPI-SC imports. Optional.
- `-DUSE_MTI_CIN`
Enables support of C++ standard input *cin*. Optional.
- `<CPP linker options>`
Any normal C++ compiler option can be used, with the exception of the `-o` option. You can specify arguments for all **sccom** compiles by editing the **CppOptions** variable in the *modelsim.ini* file.
- `-cpppath <filename>`
Specifies the location of a g++ executable other than the default g++ compiler installed with ModelSim. Optional. Overrides the **CppPath** variable in the *modelsim.ini* file.
- `-dumpscvext <filename>`
Generates SystemC verification (SCV) extensions for any given object type. For this argument, `<filename>` is a C++ (.cpp) file that contains global variable definition for each type and includes the header file containing definitions for these types. Optional.
- `-dpilib <libname>`
Specifies SystemVerilog libraries for SystemC DPI (Direct Programming Interface). Optional. Only one library can be specified per each `-dpilib` argument. See section [SystemC Procedural Interface to SystemVerilog](#).
- `-error <msg_number> [,<msg_number>,...]`
Changes the severity level of the specified message(s) to "error." Optional. Edit the `error` variable in the *modelsim.ini* file to set a permanent default. Refer to [“Changing Message Severity Level”](#) for more information.
- `-f <filename>`
Specifies an argument file with more command-line arguments. Optional. Allows complex argument strings to be reused without retyping. Nesting of `-f` options is allowed.
Refer to the section ["Argument Files"](#) for more information.
- `-fatal <msg_number> [,<msg_number>,...]`
Changes the severity level of the specified message(s) to "fatal." Optional. Edit the `fatal` variable in the *modelsim.ini* file to set a permanent default. Refer to [“Changing Message Severity Level”](#) for more information.

- -help
Displays options and arguments for this command. Optional.
- -incr
Enables automatic incremental compilation so that only changed files are compiled. Optional. A changed file is re-compiled in the following cases:
 - Its pre-processor output is different from the last time it was successfully compiled. This includes changes in included header files and to the source code itself.

Note

Pre-processor output is used because it prevents compilation on a file with the following types of changes:

- Access or modification time (touch)
 - Changes to comments—except changes to the source code that affect line numbers (such as adding a comment line) will cause all affected files to be recompiled. This occurs to keep debug information current so that ModelSim can trace back to the correct areas of the source code.
-
- You invoke sccom with a different set of command-line options that have an impact on the gcc command line. Preserving all settings for the gcc command ensures that ModelSim re-compiles source files when a different version of gcc is used or when a platform changes.

- -lib <library_pathname>
Specifies the default working library where the SystemC linker can find the object files for compiled SystemC modules. Use only with sccom -link. Optional.
- -libshared <library_pathname>
(UNIX, Linux only) Specifies the library location for the intermediate SystemC library specified with sccom -linkshared. This argument cannot be used with sccom -link. Optional.
- -link

Performs the final link of all previously compiled SystemC source code. Required before running simulation. You must specify any dependent *.a* or *.o* before the *.a* or *.o* on which it depends. Two types of dependencies are possible, and where you place the -link argument is different based on which type of dependency the files have.

If your archive or object is dependent on the *.o* files created by sccom (that is, your code references symbols in the generated SystemC *.o* files), then you must specify the -link argument after the list of files, as follows:

```
sccom a.o b.o libtemp.a -linkz
```

Functionally, the order of the C++ linker command and argument looks like this:

```
ld a.o b.o libtemp.a <internal list of SC .o files> libsystemc.a
```

However, if the *.o* files created by **sccom** are dependent on the object or archive you provided, then you must place the **-link** argument before the object files or archive:

```
sccom -link a.o b.o libtemp.a
```

In this case, the "functional" command and argument order look like this:

```
ld <internal list of SC .o files> libsystemc.a a.o b.o libtemp.a
```

- **-linkshared**
(UNIX, Linux only) Creates an intermediate SystemC shared library, which allows you to create multiple SystemC shared libraries (*systemc.so*). This argument cannot be used with **sccom -link**. Optional.
- **-log <logfile>**
Specifies the logfile in which to collect output. Optional. Related *modelsim.ini* variable is *SccomLogfile*.
- **-modelsimini <ini_filepath>**
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-nodbsym**
Disables the generation of symbols for the debugging database in the library, which allows source annotation.
- **-nodebug**
Disables the creation of a debug database for modules defined in those source files. Optional. Do not use this argument with any files containing the `SC_MODULE_EXPORT()` macro. Using **-nodebug** can significantly reduce the compile time for a design, which is useful when running designs in regression mode. Refer to "[Custom Debugging of SystemC Channels and Variables](#)" for more information.
- **-nologo**
Disables the startup banner. Optional.
- **-note <msg_number> [,<msg_number>,...]**
Changes the severity level of the specified message(s) to "note." Optional. Edit the *note* variable in the *modelsim.ini* file to set a permanent default. Refer to "[Changing Message Severity Level](#)" for more information.
- **-scms**
Includes the SystemC master slave library. Optional. If you specify this argument when compiling your C code with **sccom**, you must also specify it when linking the object files with **sccom -link**.

- **-scv**
Includes the SystemC verification library. Optional. If you specify this argument when compiling your C code with **sccom**, you must also specify it when linking the object files with **sccom -link**. Related *modelsim.ini* variable is UseScv.
- **-scversion**
Prints out the version of the SystemC verification library used. Optional.
- **-suppress <msg_number> [,<msg_number>,...]**
Prevents the specified message(s) from displaying. Optional. You cannot suppress Fatal or Internal messages. Edit the **suppress** variable in the *modelsim.ini* file to set a permanent default.
- **-vv**
Prints all subprocess invocation information. Optional. An example is the call to **gcc** along with the command-line arguments.
- **-verbose**
Prints the name of each *sc_module* encountered during compilation. Optional. Related *modelsim.ini* variable *SccomVerbose*.
- **-version**
Displays the version of **sccom** used to compile the design. Optional.
- **-warning <msg_number> [,<msg_number>,...]**
Changes the severity level of the specified message(s) to "warning." Optional. Edit the **warning** variable in the *modelsim.ini* file to set a permanent default.
- **-work <library_name>**
For the compiler — Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional; by default, the compiled object files (.so) are added to the **work** library. The specified pathname overrides the pathname specified for *work* in the project file.
For the linker — Specifies a logical name or pathname of a library where the final linked object file (.so) is to be stored. Optional; by default, the linked object files are added to the **work** library.
- **-x c | c++**
Specifies the language (C or C++) of a source file being compiled when the filename extension does not match that source. Optional. For example, if *myfile.cpp* contained C source, you would enter **sccom -x c myfile.cpp**. If you use this argument, you cannot combine C and C++ files with the same **sccom** command.

- <filename>

Specifies the name of a file containing the SystemC/C++ source to be compiled. Required. You can enter multiple filenames separated by spaces; you can also use wildcards to specify multiple filenames (such as **.cpp*).

Examples

- Compile *example.cpp* with debugging information.

```
sccom -g example.cpp
```

- Link *example.o*.

```
sccom -link
```

- Use two **sccom** commands - the first to compile *a.cpp* into library *LIB_A*, and the second to compile *b.cpp* into *LIB_B*. (*a.cpp* defines a module, *TOP_A*, and *b.cpp* defines a module, *TOP_B*.) Run **sccom** again to link and compile the compiled object files created in those two libraries into a third shared library, *LIB_C*. Run **vsim** using **-sclib**, which is required in order to point to the location of the shared library.

```
vlib /path/to/LIB_A  
vmap LIB_A /path/to/LIB_A  
vlib /path/to/LIB_B  
vmap LIB_B /path/to/LIB_B
```

```
sccom -work LIB_A a.cpp -> a.o created in /path/to/LIB_A  
sccom -work LIB_B b.cpp -> b.o created in /path/to/LIB_B
```

At this point you have the option to create the SystemC library in *LIB_A* or *LIB_B* or in a totally new library *LIB_C*.

Include all objects from *LIB_B* and *LIB_A* and create a *.so* (shared object) in *LIB_A*:

```
sccom -link -work LIB_A -lib LIB_B
```

Include all objects created in *LIB_A* and *LIB_B* and *LIB_C* and create a *.so* in *LIB_C*.

```
sccom -link -work LIB_C -lib LIB_A -lib LIB_B
```

If the shared object is not compiled in the same library as the top-level design unit, the *.so* library has to be specified using the **-sclib** switch with the **vsim** command. The **-lib** switch tells the simulator where the top-level module is compiled. The **vsim** command also has a **-L** switch that allows you to specify the library where lower level modules can be found. For example:

```
vsim -sclib LIB_C -lib LIB_A TOP_A
```

loads a SystemC shared library from *LIB_C*. The top module *TOP_A*, is compiled in *LIB_A*.

```
vsim -lib LIB_A TOP_A
```

loads the SystemC shared library from *LIB_A* and the top module *TOP_A* from *LIB_A*.

```
vsim -lib LIB_A -sclib LIB_C -L LIB_B TOP_A
```

loads the SystemC shared library from *LIB_C*. The top level module, *TOP_A*, was compiled in *LIB_A*. *TOP_B*, which is instantiated in some hierarchy, can be found in *LIB_B*.

The **vsim** command can accept multiple **-L** switches, but it takes only one **-lib** switch. The **-lib** switch is for top-level modules and **-L** is for lower modules.

The **-sclib** switch specifies where the SystemC shared library was created.

- Compile the SystemC code with an include directory and the compile time macro (SC_INCLUDE_FX) to compile the source with support for fixed point types. For more information, refer to “[Fixed-Point Types](#)”.

```
sccom -I/home/systemc/include -DSC_INCLUDE_FX -g a.cpp b.cpp
```

- Compile with the g++ -O2 optimization argument.

```
sccom -O2 a.cpp
```

- Link in the library *libmylib.a* when creating the *.so* file. The **-L**, a **gcc** argument, specifies the search path for the libraries.

```
sccom -L home/libs/ -l mylib -link
```

See also

“[SystemC Simulation](#)”, [sccgenmod](#), [vdel -allsystemc](#)

scgenmod

Once a Verilog or VHDL module is compiled into a library, you can use the **scgenmod** command to write its equivalent SystemC foreign module declaration to standard output.

Optional **-map** argument allow you to appropriately generate `sc_bit`, `sc_bv`, or resolved port types; `sc_logic` and `sc_lv` port types are generated by default.

Syntax

```
scgenmod [-help] [-lib <library_name>] [-map "<hdl_type>=<sc_type>"]  
        [-modelsimini <ini_filepath>] [-createtemplate] <module_name>
```

Arguments

- **-createtemplate**
Creates a class template declaration of a foreign module with integer template arguments corresponding to the integer parameter/generic defined in the VHDL or Verilog module. Ports in VHDL and Verilog modules instantiated from SystemC can now have their range specified in terms of integer parameters/generics. Such port ranges will be specified in terms of the template arguments of the foreign module.
- **-help**
Displays the command's options and arguments. Optional.
- **-lib <library_name>**
Specifies the pathname of the working library. If not specified, the default library **work** is used. Optional.
- **-map "<hdl_type>=<sc_type>"**
Specifies the user defined type mappings between either SystemVerilog or VHDL and SystemC types. `<hdl_type>` is the supported SystemVerilog or VHDL types. `<sc_type>` is the supported SystemC types. No spaces are allowed. Optional.
 - SystemVerilog supported types:
scalar (Verilog scalar), vector (Verilog vector), bit, logic, reg, bitvector (signed/unsigned, packed/unpacked bit vector), logicvector (signed/unsigned, packed/unpacked logic vector), regvector (signed/unsigned, packed/unpacked logic vector), integer, integer unsigned, int, int unsigned, shortint, shortint unsigned, longint, longint unsigned, byte, byte unsigned, struct (including fields with multi-dimensional arrays)

Note



You can map Verilog vectors to the following native C++ integer types:
short, int, long long.

- VHDL supported types:
bit, bit_vector, enum, record (including nested records and fields with multi-dimensional arrays), std_logic, std_logic_vector, vl_logic, vl_logic_vector

- SystemC supported types:
bool, sc_bit, sc_logic, sc_bv, sc_lv, short, unsigned short, int, unsigned int, long, unsigned long, long long, unsigned long long, sc_int, sc_signed, sc_unsigned, sc_bigint, sc_biguint, char, unsigned char

Note

VHDL/SystemVerilog ports of type multi-dimensional array get converted to SystemC signal arrays in the generated foreign module declaration.

- `-modelsimini <ini_filepath>`
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- `<module_name>`
Specifies the name of the Verilog/VHDL module to be accessed. Required.

Obsolete for the Current Release

The following options have become obsolete. As of the 6.3 release, the types are mapped automatically according to the specifications given to the **-map** argument.

- `-sc_bit`
Causes **scgenmod** to generate `sc_bit` scalar port types. Obsolete. See the **-map** argument.
- `-bool`
Causes **scgenmod** to generate boolean scalar port types. Obsolete. See the **-map** argument.
- `-sc_logic`
Causes **scgenmod** to generate `sc_logic` scalar port types. Obsolete. See the **-map** argument.
- `-sc_resolved`
Causes **scgenmod** to generate resolved scalar port types. Obsolete. See the **-map** argument.
- `-sc_bv`
Causes **scgenmod** to generate `sc_bv<N>` vector port types. Obsolete. See the **-map** argument.
- `-sc_int`
Causes **scgenmod** to generate `sc_int<N>` vector port types. Obsolete. See the **-map** argument.
- `-sc_lv`
Causes **scgenmod** to generate `sc_lv<N>` vector port types. Obsolete. See the **-map** argument.

- `-sc_rv`
Causes **scgenmod** to generate resolved vector port types. Obsolete. See the **-map** argument.
- `-sc_uint`
Causes **scgenmod** to generate `sc_uint<N>` port types. Obsolete. See the **-map** argument.

Examples

- This example uses a Verilog module that is compiled into the work library. The module begins as Verilog source code:

```
module vcounter (clock, topcount, count);
    input clock;
    input topcount;
    output count;
    reg count;
    ...
endmodule
```

- After compiling using `vlog`, you invoke **scgenmod** on the compiled module with the following command:

```
scgenmod vcounter
```

The SystemC foreign module declaration for the above Verilog module is:

```
class vcounter : public sc_foreign_module
{
    public:
        sc_in<sc_logic> clock;
        sc_in<sc_logic> topcount;
        sc_out<sc_logic> count;
        vcounter(sc_module_name nm),
            : sc_foreign_module(nm, hdl_name),
              clock("clock"),
              topcount("topcount"),
              count("count")
            {elaborate_foreign_module(hdl_hame);}
        ~vcounter()
        {}
};
```

See also

“SystemC Simulation”, `scom`

sdfcom

This command compiles the specified SDF file. Annotation of compiled SDF files can dramatically improve performance (compared to annotating the ASCII version of the same) in cases where the same SDF file is used for multiple simulation runs.

The compiled SDF file is annotated so that it is compatible with the **vsim -v2k_int_delays** command (that is, the annotator operates as if the **vsim -v2k_int_delays** command has been given).

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
sdfcom [-delayscale <value>] [-maxdelays] [-mindelays] [-nocompress] [-typdelays]  
      <source_file> <output_file>
```

Arguments

- **-delayscale <value>**
(optional) Scales delays by the specified value.
<value> —
If you use this argument during SDF compilation, do *not* use the scaling option when reading in the SDF file with **vsim**. If you do so, the delays will be scaled twice.
- **-maxdelays**
(optional) Selects maximum delays from SDF delay values of the form (min:typ:max).
- **-mindelays**
(optional) Selects minimum delays from SDF delay values of the form (min:typ:max).
- **-nocompress**
(optional) Produces a compiled file that is not compressed with **gzip**. By default the compiled file is compressed with **gzip** (even though the resulting file does not have the usual **.gz** extension).
- **-typdelays**
(optional) Selects typical delays from SDF delay values of the form (min:typ:max). Default.
- **<source_file>**
(required) Specifies the SDF file to compile. Must be specified immediately in front of the **<output_file>** argument to the **sdfcom** command.
- **<output_file>**
(required) Specifies the name for the compiled SDF file. Must be specified as the final argument to the **sdfcom** command.

Related Topics

- [vsim -sdftyp](#)
- [Compiling SDF Files](#)

search

This command searches the specified window for one or more objects matching the specified pattern(s). The search can be continued using the [next](#) command.

The search starts at the object currently selected, if any; otherwise it starts at the window top. The default action is to search downward until the first match, then move the selection to the object found, and return the index of the object found.

Returns the index of a single match, or a list of matching indexes. Returns nothing if no matches are found.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
search <window_name> [-window <wname>] [-all] [-field <n>] [-toggle]
  [-forward | -backward] [-wrap | -nowrap] [-exact] [-regexp] [-nocase] [-count <n>]
  <pattern>
```

Arguments for all windows

- <window_name>
(required) Specifies the window in which to search. Can be one of:
list, locals, objects, process, source, structure, or wave
or a unique abbreviation. Must be specified as the first argument to the **search** command.
- -window <wname>
(optional) Specifies a particular window when multiple instances of the window exist. If no window is specified the default window is used; the default window is determined by the most recent invocation of the [view](#) command.

<wname> — Specifies the instance of the window to use (for example, wave2).
- -forward
(optional) Search in the forward direction. (default)
- -backward
(optional) Search in the reverse direction.
- <pattern>
(required) String or glob-style wildcard pattern. Must be specified as the final argument to the **search** command.

Arguments, for all EXCEPT the Source window

- -all
(optional) Finds all matches and returns a list of the indexes of all objects that match.

- -field <n>
(optional) Selects different fields to test, depending on the window type:

Table 2-6. Field Arguments for Window Searches

Window	n=1	n=2	n=3	default
structure	instance	entity/module	architecture	instance
signals	name	-	cur. value	name
process	status	process label	fullpath	fullpath
variables	name	-	cur. value	name
wave	name	-	cur. value	name
list	label	fullname	-	label

Default behavior for the List window is to attempt to match the label and if that fails, try to match the full signal name.

- -toggle
(optional) Adds objects found to the selection. Does not do an initial clear selection. Otherwise deselects all and selects only one object.
- -wrap
(optional) Specifies that the search continue from the top of the window after reaching the bottom. (default)
- -nowrap
(optional) Specifies that the search stop at the bottom of the window and not continue searching from the top.

Arguments, Source window only

- -exact
(optional) Search for an exact match.
- -regexp
(optional) Use the pattern as a Tcl regular expression.
- -nocase
(optional) Ignore case where the default is to use case.
- -count <n>
(optional) Search for the nth match where the default is to search for the first match.

Description

With the **-all** option, the entire window is searched, the last object matching the pattern is selected, and a Tcl list of all corresponding indexes is returned.

With the **-toggle** option, objects found are selected in addition to the current selection.

For the List window, the search is done on the names of the objects listed, that is, across the header. To search for values of objects in the List window, use the [down](#) and [up](#) commands. Likewise, in the Wave window, the search is done on object names and values in the values column. To search for object values in the waveform pane of the Wave window, use the [right](#) and the [left](#) commands. You can also select **Edit > Search** in both windows.

Related Topics

- [find](#)
- [next](#)
- [view](#)

searchlog

This command searches one or more of the currently open logfiles for a specified condition.

It can be used to search for rising or falling edges, for signals equal to a specified value, or for when a generalized expression becomes true.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
searchlog [-count <n>] [-deltas] [-endtime <time> [<unit>]] [-env <path>] [-event <time>]
  [-expr {<expr>}] [-reverse] [-rising | -falling | -anyedge] [-startDelta <num>]
  [-value <string>] <startTime> [<unit>] <pattern>
```

Description

If at least one match is found, it returns the time (and, optionally, delta) at which the last match occurred and the number of matches found, in a Tcl list:

```
{ {<time>} <matchCount> }
```

where **<time>** is in the format **<number> <unit>**. If the **-deltas** option is specified, the delta of the last match is also returned:

```
{ {<time>} <delta> <matchCount> }
```

If no matches are found, a `TCL_ERROR` is returned. If one or more matches are found, but less than the number requested, it is not considered an error condition, and the time of the farthest match is returned, with the count of the matches found.

Arguments

- **-count <n>**
(optional) Specifies to search for the nth occurrence of the match condition.
<n> — Any positive integer.
- **-deltas**
(optional) Indicates to test for a match on simulation delta cycles. Otherwise, matches are only tested for at the end of each simulation time step.
- **-endtime <time> [<unit>]**
(optional) Specifies the simulation time at which to start the search. By default there is no end time specified.
<time> — Specified as an integer or decimal number. Current simulation units are the default unless specifying **<unit>**.
<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid VHDL time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- **-env <path>**
(optional) Indicates to test for a match on simulation delta cycles. Otherwise, matches are only tested for at the end of each simulation time step. **ath>** — A path to a design region. Wildcards not allowed.
- **-event <time>**
(optional) Indicates to test for a match on a simulation event. Otherwise, matches are only tested for at the end of each simulation time step.
- **-expr {<expr>}**
(optional) Specifies a search for a general expression of signal values and simulation time. **searchlog** will search until the expression evaluates to true.
{<expr>} — An expression that evaluates to a boolean true. See [GUI_expression_format](#) for the format of the expression.
- **-reverse**
(optional) Specifies to search backwards in time from **<startTime>**.
- **-rising**
(optional) Specifies a search for rising edge on a scalar signal. This option is ignored for compound signals.
- **-falling**
(optional) Specifies a search for falling edge on a scalar signal. This option is ignored for compound signals.
- **-anyedge**
(optional) Specifies a search for a rising or falling edge on a scalar signal. This option is ignored for compound signals. (default)
- **-startDelta <num>**
(optional) Indicates a simulation delta cycle on which to start.
<num> — Any positive integer.
- **-value <string>**
(optional) Specifies a match of a single scalar or compound signal against a specified string.
<string> — Specifies a string to be matched.
- **<startTime> [<unit>]**
(required) Specifies the simulation time at which to start the search. The time is specified as an integer or decimal number. Must be placed immediately before the **<pattern>** argument.
<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid VHDL time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces (**{}**).

- `<pattern>`
(Required unless the **-expr** argument is used.) Specifies one or more signal names or wildcard patterns of signal names to search on. Must be specified as the final argument to the **searchlog** command.

Related Topics

- [virtual signal](#)
- [virtual log](#)
- [virtual nolog](#)
- [GUI_expression_format](#)

see

This command displays the specified number of source file lines around the current execution line and places a marker to indicate the current execution line. If specified without arguments, five lines will be displayed before and four lines after.

Syntax

```
see [<n> | <pre> <post>]
```

Arguments

- <n>
(optional) Designates the number of lines to display before and after the current execution line.
- <pre>
(optional) Designates the number of lines to display before the current execution line.
- <post>
(optional) Designates the number of lines to display after the current execution line.

Example

- Display 2 lines before and 5 lines after the current execution line.

see 2 5

Returns:

```
# 92 :  
# 93 :           if (verbose) $display("Read/Write test done");  
# ->94 :           $stop(1);  
# 95 :           end  
# 96 :       end  
# 97 :  
# 98 :       or2 i1 (  
# 99 :           .y(t_set),
```

seetime

This command scrolls the List or Wave window to make the specified time visible.

For the List window, a delta can be optionally specified as well.

Returns the specified time and unit.

Arguments to this command are order dependent. Please read the argument descriptions for more information. “<time> [<unit>]” must be specified as the final argument to the **seetime** command.

Syntax

```
seetime list | wave [-window <wname>] [-select] [-delta <num>] [-event] <time> [<unit>]
```

Arguments

- list | wave
(required) Controls which type of window to target. Must be specified as the first argument to the **seetime** command.
 - list — Target a list window.
 - wave — Target a wave window.
- -window <wname>
(optional) Specifies that a window other than the current default Wave or List window is used. The default is to use the current default Wave or List window when the **-window** argument is not specified. Use the [view](#) command to change the default window.
 - <wname> — Specifies an instance of the Wave or List window that is not the current default.
- -select
(optional) Also moves the active cursor or marker to the specified time (and optionally, delta). Otherwise, the window is only scrolled.
- -delta <num>
(optional) For the List window when deltas are not collapsed, this option specifies a delta. Otherwise, delta 0 is selected.
 - <num> — Any positive integer.
- -event
(optional) List or Wave window scrolls to the specified event.
- <time> [<unit>]
<time> — (required) Specified as an integer or decimal number. Current simulation units are the default unless specifying <unit>. Must be specified as the final argument to the **seetime** command.

<unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid VHDL time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).

setenv

This command changes or reports the current value of an environment variable. The setting is valid only for the current ModelSim session.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
setenv <varname> [<value>]
```

Arguments

- <varname>
(required) The name of the environment variable you wish to set or check. Must be specified as the first argument to the **setenv** command.
- <value>
(optional) The new value for <varname>. If you do not specify a value, ModelSim reports the variable's current value.

Related Topics

- [unsetenv](#)
- [printenv](#)

shift

This command shifts macro parameter values left one place, so that the value of parameter \(\$2 is assigned to parameter \(\$1, the value of parameter \(\$3 is assigned to \(\$2, and so on. The previous value of \(\$1 is discarded.

The **shift** command and macro parameters are used in macro files. If a macro file requires more than nine parameters, they can be accessed using the **shift** command.

To determine the current number of macro parameters, use the [argc](#) variable.

Syntax

shift

Arguments

- None

Description

For a macro file containing nine macro parameters defined as \$1 to \$9, one **shift** command shifts all parameter values one place to the left. If more than nine parameters are named, the value of the tenth parameter becomes the value of \$9 and can be accessed from within the macro file.

Related Topics

- [do](#)

show

This command lists HDL objects and subregions visible from the current environment.

The objects listed include:

- **VHDL** — signals, processes, constants, variables, and instances.
- **Verilog** — nets, registers, tasks, functions, instances, variables, and memories.

When using **C Debug**, **show** displays the names and types of the local variables and arguments of the current C function.

The **show** command returns formatted results to stdout. To eliminate formatting (to use the output in a Tcl script), use the **Show** form of the command instead.

Syntax

```
show [-all] [<pathname>]
```

Arguments

- **-all**
(optional) Displays all names at and below the specified path recursively.
- **<pathname>**
(optional) Specifies the pathname of the environment for which you want the objects and subregions to be listed; if omitted, the current environment is assumed.

Examples

- List the names of all the objects and subregion environments visible in the current environment.
show
- List the names of all the objects and subregions visible in the environment named /uut.
show /uut
- List the names of all the objects and subregions visible in the environment named sub_region which is directly visible in the current environment.
show sub_region

Related Topics

- [environment](#)
- [find](#)

simstats

This command returns performance-related statistics about elaboration and simulation. The statistics measure the simulation kernel process (vsimk) for a single invocation of vsim. If you invoke vsim a second time, or restart the simulation, the current statistics are discarded and new values are collected.

If executed without arguments, the command returns a list of pairs similar to the following:

```
{{elab memory} 0} {{elab working set} 7245824} {{elab time} 0.942645}  
{{elab cpu time} 0.190274} {{elab context} 0} {{elab page faults} 1549}  
{memory 0} {{working set} 0} {time 0} {{cpu time} 0} {context 0}  
{{page faults} 0}
```

The elaboration statistics are measured one time at the end of elaboration. The simulation memory statistics are measured at the time you invoke **simstats**. The simulation time statistics are updated at the end of each **run** command. See the arguments below for descriptions of each statistic.

Units for time values are in seconds. Units for memory values vary by platform:

- For SunOS and Linux, the memory size is reported in Kbytes
- For Windows, the memory size is reported in bytes.

Some of the values may not be available on all platforms and other values may be approximates. Different operating systems report these numbers differently.

Syntax

```
simstats [memory | working | time | cpu | context | faults]
```

Arguments

- **memory**
(optional) Returns the amount of virtual memory that the OS has allocated for vsimk.
- **working**
(optional) Returns the portion of allocated virtual memory that is currently being used by vsimk. If this number exceeds the actual memory size, you will encounter performance degradation.
- **time**
(optional) Returns the cumulative “wall clock” time of all run commands.
- **cpu**
(optional) Returns the cumulative processor time of all run commands. Processor time differs from wall clock time in that processor time is only counted when the cpu is actually running vsimk. If vsimk is swapped out for another process, cpu time does not increase.

- context
(optional) Returns the number of context swaps (vsimk being swapped out for another process) that occurred during all run commands.
- faults
(optional) Returns the number of page faults that occurred during all run commands.

status

This command lists summary information about currently interrupted macros.

If invoked without arguments, the command lists the filename of each interrupted macro, the line number at which it was interrupted, and prints the command itself. It also displays any [onbreak](#) or [onerror](#) commands that have been defined for each interrupted macro.

Syntax

```
status [file | line]
```

Arguments

- **file**
(optional) Reports the file pathname of the current macro.
- **line**
(optional) Reports the line number of the current macro.

Examples

The transcript below contains examples of [resume](#), and **status** commands.

```
VSIM(paused)> status
# Macro resume_test.do at line 3 (Current macro)
#   command executing: "pause"
#   is Interrupted
#   ONBREAK commands: "resume"
# Macro startup.do at line 34
#   command executing: "run 1000"
#   processing BREAKPOINT
#   is Interrupted
#   ONBREAK commands: "resume"
VSIM(paused)> resume
# Resuming execution of macro resume_test.do at line 4
```

Related Topics

- [abort](#)
- [do](#)
- [pause](#)
- [resume](#)

step

The **step** command is an alias for the run command with the -step switch.

Refer to the [run](#) command for a complete syntax description.

stop

This command is used with the [when](#) command to stop simulation in batch files.

The **stop** command has the same effect as hitting a breakpoint. The **stop** command may be placed anywhere within the body of the when command.

Syntax

```
stop [-sync]
```

Arguments

- `-sync`
(optional) Stops the currently running simulation at the next time step.

Description

Use [run -continue](#) to continue the simulation run, or the [resume](#) command to continue macro execution. If you want macro execution to resume automatically, put the **resume** command at the top of your macro file:

```
onbreak {resume}
```

Note



If you want to stop the simulation using a [when](#) command, you must use a stop command within your when statement. DO NOT use an [exit](#) command or a [quit](#) command. The **stop** command acts like a breakpoint at the time it is evaluated.

Related Topics

- [bp](#)
- [resume](#)
- [run](#)
- [when](#)

suppress

This command prevents one or more specified messages from displaying. You cannot suppress Fatal or Internal messages. The **suppress** command used without arguments returns the message numbers of all suppressed messages.

Edit the **suppress** variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

Syntax

```
suppress [-clear <msg_number>[,<msg_number>,...]] [<msg_number>[,<msg_number>,...]]  
        [<code_string>[, <code_string>,...]]
```

Arguments

- `-clear <msg_number>[,<msg_number>,...]`
(optional) Clears suppression of one or more messages identified by message number.
`<msg_number>` — A number identifying the message. Multiple message numbers are specified as a comma separated list.
- `<msg_number>[,<msg_number>,...]`
(optional) A number identifying the message to be suppressed. Multiple message numbers are specified as a comma separated list.
- `<code_string>[, <code_string>,...]`
(optional) A string identifier of the message to be suppressed. Disables warning messages in the category specified by `<CODE>`. Warnings that can be disabled include the `<CODE>` name in square brackets in the warning message.

Examples

- Return the message numbers of all suppressed messages:
suppress
- Suppress messages by message number:
suppress 8241,8242,8243,8446,8447
- Suppress messages by numbers and code categories:
suppress 8241,TFMPC,CNNODP,8446,8447
- Clear message suppression for the designated messages:
suppress -clear 8241,8242
- Return the message numbers of all suppressed messages and clear suppression for all:
suppress -clear [suppress]

tb

This (traceback) command (traceback) displays a stack trace for the current process in the Transcript window. This lists the sequence of HDL function calls that have been entered to arrive at the current state for the active process.

If you are using [C Debug](#), **tb** displays a stack trace of the C call stack.

Syntax

```
tb [<#_of_levels>]
```

Arguments

- `<#_of_levels>`
(optional) Specifies the number of call frames in the C stack to display. If you do not specify a level, the entire C stack is displayed. This argument is available only for C Debug.

tcheck_set

This command works in tandem with [tcheck_status](#) to report on and enable/disable individual timing checks. **tcheck_set** modifies either a check's reporting or X-generation status, and reports the new setting in the Transcript window.

Disabling a timing check's reporting prevents generation of associated violation messages. For Verilog modules this means ModelSim disables message reporting. For VHDL design units this means ModelSim sets the MsgOn parameter in a VITAL timing check procedure (TCP) to FALSE. Disabling a timing check's X generation removes a timing check's ability to affect the outputs of the simulation. For Verilog modules, this means ModelSim toggles the timing check's notifier. For VHDL design units, this means ModelSim sets the Xon parameter in a VITAL TCP to FALSE.

Note that the **tcheck_set** command *does not* override the effects of invoking [vlog](#) or [vsim](#) with the **+nospecify**, **+notimingchecks**, or **+no_neg_tchk** arguments. Using **tcheck_set** can override the effects of invoking [vsim](#) with the **+no_notifier**, **+no_tchk_msg**, **-g**, or **-G** arguments. These latter arguments establish initial values for the simulation, and those values can be modified by **tcheck_set**.

Note



If you have compiled a design using [vlog](#) -nodebug, you can turn off timing checks with **tcheck_set** however, no record of the new setting is reported to the Transcript window. Also, [tcheck_status](#) will not report the status of a timing check set with **tcheck_set**.

Keep in mind the following if you are using VHDL VITAL:

- VITAL does not provide the granularity to set individual period or width checks. These checks are part of a single VITAL TCP, and **tcheck_set** toggles MsgOn and Xon for all checks in the TCP. See "Examples" below for further information.
- If an instance is not Level-1 optimized, you cannot set values for individual TCPs. You can set values only for the entire instance. **tcheck_status** reports "ALL" for instances that aren't Level-1 optimized. See "Examples" below for further information.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

```
tcheck_set [-quiet] <instance> [{-m | -n}] [-r [-v]]  
          [<Stat> | <MsgStat> <XStat>]
```

Arguments

- <instance>
(required) Specifies the instance for which you want to change the reporting or X-generation status.

- -m | -n
(optional) Specifies whether *instance* is a module/entity or a net.
 - m — Indicates that the *instance* is a module or entity and that tcheck_set will operate on all instances of the specified module type.
 - n — Indicates that the *instance* is a hierarchical pathname to a net and that tcheck_set will operate on all instances connected to the specified net.
- -quiet
(optional) Suppresses printing the new setting to the Transcript window. If you use this switch, it must precede the *instance* argument. You can specify -q as an alias of this switch.
- -r [-v]
(optional) Attempts to change all checks on *instance* and all instances below (recursive). By default, the recursively altered instances are not printed to the transcript.
 - v — (optional) Specifies that recursively altered instances should be printed to the transcript.
- <tcheck> | ALL
(optional) Specifies which checks should be changed.
 - <tcheck> — (optional) Specifies a specific timing check to change. You can specify either:
 - The integer that is assigned to each timing check (and reported via **tcheck_status**). Note that the integer number may change between library compiles.
 - The full timing check name enclosed in double quotes, such as:
“(WIDTH (negedge CLK))”
 - The type of timing check, with out any port identifiers, enclosed in double quotes. This acts as a form of wildcard specification, such as:
“(WIDTH)”
which instructs tcheck_set to operate on all timing checks with the type “WIDTH” in *instance*.
 - ALL — Attempts to change all checks in *instance*. (default)
- <Stat>
(optional) Enables/disables both violation message and X generation reporting for the specified timing check(s).
 - ON — Enable both violation and X generation reporting.
 - OFF — Disable both violation and X generation reporting.

- `<MsgStat> <XStat>`
(optional) Individually controls both violation message and X generation reporting for the specified timing check(s). Both variables, `<MsgStat>` and `<XStat>`, are specified together. `<MsgStat>` must be specified before `<XStat>`. For example,
ON OFF— Enable violation message reporting and disable X generation reporting.
OFF ON — Disable violation message reporting and enable X generation reporting.

Examples

- Turn off message reporting and X generation for the "(WIDTH (negedge CLK))" check in instance `top.y1.u2`. These examples assume that your PathSeparator variable is set to "." rather than the default "/".

```
tcheck_set top.y1.u2 "( WIDTH (negedge CLK) )" OFF
```

Returns the following output in the Transcript window:

```
#0 ( WIDTH (negedge CLK) ) MsgOff XOff
```

- Turn off message reporting for timing check number 1 in instance `top.y1.u2`.

```
tcheck_set top.y1.u2 1 OFF ON
```

Returns the following output in the Transcript window:

```
#1 ( WIDTH (posedge CLK) ) MsgOff XOn  
VSIM 2> tcheck_status dff1  
# 1 ( PERIOD CLK ) MsgOn, XOn  
# ( WIDTH (posedge CLK) ) MsgOn, XOn  
# ( WIDTH (negedge CLK) ) MsgOn, XOn  
VSIM 3> tcheck_set dff1 "( WIDTH (posedge CLK) )" off on  
# 1 ( PERIOD CLK ) MsgOff, XOn  
# ( WIDTH (posedge CLK) ) MsgOff, XOn  
# ( WIDTH (negedge CLK) ) MsgOff, XOn
```

Show how period and hold checks work with VHDL VITAL. In this case, specifying "off on" for (WIDTH (posedge CLK)) also sets (PERIOD CLK) and (WIDTH (negedge CLK)) to the same values.

```
VSIM 3> tcheck_status dff5  
# ALL MsgOn XOn  
VSIM 4> tcheck_set dff5 on off  
# ALL MsgOn XOff
```

Instance `dff5` is from an unaccelerated model so **tcheck_set** can only toggle message reporting and X generation for all checks on the instance.

- Turn off message reporting and X generation on all WIDTH statements on modules of type "mymod".

```
tcheck_set mymod -m "(WIDTH)" OFF
```

- Recursively turn on message reporting and turn off X generation on all SETUP statements on instances driven by the net top.clk.

```
tcheck_set top.clk -r -n "(SETUP)" ON OFF
```

Related Topics

- [tcheck_status](#)
- [Compiling and Simulating with Accelerated VITAL Packages](#)
- [SDF Timing Annotation](#)
- [Disabling Timing Checks](#)
- [vsim -g](#)
- [vsim -G](#)
- [vsim +no_notifier](#)
- [vsim +no_tchk_msg](#)
- [vsim +nospecify](#)
- [vsim +no_neg_tchk](#)
- [vsim +notimingchecks](#)

tcheck_status

This command works in tandem with `tcheck_set` to report on and enable/disable individual timing checks. `tcheck_status` prints in the Transcript window the current status of all timing checks in the instance or a specific timing check specified with the optional `<tcheck>` argument.

You can prevent the generation of associated violation messages by disabling the reporting of a timing check. For Verilog modules this means ModelSim disables message reporting. For VHDL design units this means ModelSim sets the `MsgOn` parameter in a VITAL timing check procedure (TCP) to `FALSE`. Disabling a timing check's X generation removes a timing check's ability to affect the outputs of the simulation. For Verilog modules this means ModelSim toggles the timing check's notifier. For VHDL design units this means ModelSim sets the `Xon` parameter in a VITAL TCP to `FALSE`.

Syntax

```
tcheck_status [-lines] [-m | -n] [-r] <instance> [<tcheck>]
```

You must specify the arguments in the order shown.

Arguments

- `<instance>`
(required) Specifies the instance for which you want timing check status reported.
- `-m | -n`
(optional) Specifies whether *instance* is a module/entity or a net.
 - m — Indicates that the *instance* is a module or entity and that `tcheck_status` will operate on all instances of the specified module type.
 - n — Indicates that the *instance* is a hierarchical pathname to a net and that `tcheck_status` will operate on all instances connected to the specified net.
- `-lines`
(optional) Specifies that the HDL source file and line numbers of the check(s) be displayed. Has no effect on VHDL instances. Note that line information may not always be available. You can specify `-l` as an alias for this switch.
- `-r`
(optional) Operates recursively on all instances.
- `<tcheck>` — (optional) Specifies a specific timing check to report. You can specify either:
 - The integer that is assigned to each timing check (and reported via `tcheck_status`). Note that the integer number may change between library compiles.
 - The full timing check name enclosed in double quotes, such as:
“(WIDTH (negedge CLK))”

- o The type of timing check, with out any port identifiers, enclosed in double quotes. This acts as a form of wildcard specification, such as:

“(WIDTH)”

which instructs tcheck_set to operate on all timing checks with the type “WIDTH” in *instance*.

Output

The output of **tcheck_status** is displayed in the following form:

```
#<Number> <SDF_Description> [<src_line>] <MsgStat> <XStat>
```

Table 2-7 contains a short description of each field in the output.

Table 2-7. Output Fields for tcheck_status Command

Field	Description
<Number>	an integer that can be used as shorthand to specify the check in the tcheck_status or tcheck_set commands (as the <tcheck> argument); this number can change with compiler optimizations, and you can't assume it will stay the same between library compiles
<SDF_Description>	an SDF specification of the timing check including enclosing parentheses '()'
<src_line>	the source file and line number for the timing check specification; output if you specify the -lines argument; the format of the object is <source_file_name>:<line_number>.
<MsgStat>	violation message reporting status indicator <ul style="list-style-type: none"> • MsgON/MsgOFF — violation reporting is enabled/disabled and unchangeable • MsgOn/MsgOff — violation reporting is enabled/disabled and modifiable
<XStat>	violation X generation status indicator <ul style="list-style-type: none"> • XON/XOFF — X generation is enabled/disabled and unchangeable • XOn/XOff — X generation is enabled/disabled and modifiable

Examples

- Report the timing checks for the instance “top.y1.u2”:

```
tcheck_status top.y1.u2
```

Returns:

```
#0 ( WIDTH (negedge CLK) ) MsgOn XOn
#1 ( WIDTH (posedge CLK) ) MsgOn XOn
#2 ( SETUP (negedge D) (posedge CLK) ) MsgOFF XOFF
#3 ( HOLD (posedge CLK) (negedge D) ) MsgOn XOff
```

- Report the timing checks for the instance “top.y1.u2” with line numbers:

```
tcheck_status -lines top.y1.u2 1
```

Returns:

```
#1 ( WIDTH (posedge CLK) ) 'cell.v:224' MsgOn XOn
```

- Report the timing check with the description “WIDTH (posedge CLK)”:

```
tcheck_status -lines top.y1.u2 "WIDTH (posedge CLK)"
```

Returns:

```
#1 ( WIDTH (posedge CLK) ) 'cell.v:224' MsgOn XOn
```

- Report the timing check associated with the tcheck number “1”:

```
tcheck_status -lines top.y1.u2 1
```

Returns:

```
#1 ( WIDTH (posedge CLK) ) 'cell.v:224' MsgOn XOn
```

- Report the status of all WIDTH statements on modules of type "mymod".

```
tcheck_status mymod -m "(WIDTH)"
```

- Recursively report the status of all SETUP statements on instances driven by the net top.clk.

```
tcheck_status top.clk -r -n "(SETUP)"
```

Related Topics

- [tcheck_set](#)
- [SDF Timing Annotation](#)

Time

These commands allow you to perform comparisons between, operations on, and conversions of, time values.

When [unit] is left unspecified the commands assume the current simulation time unit, as defined by the **Resolution** variable in the *modelsim.ini* file or the **vsim -t** command. Units of time (ns, us, and so forth) can be specified independently for each <time[1 | 2]> specified for most of the commands. See the description of each command and examples for more information.

Arguments to this command are order dependent. Follow the order specified in the Syntax for each command.

Syntax

eqTime <time1>[unit] <time2>[unit]

Returns a 1 (true) or 0 (false) if <time1> and <time2> are equal.

neqTime <time1>[unit] <time2>[unit]

Returns a 1 (true) or 0 (false) if <time1> and <time2> are not equal.

ltTime <time1>[unit] <time2>[unit]

Returns a 1 (true) or 0 (false) if <time1> is less than <time2>.

gtTime <time1>[unit] <time2>[unit]

Returns a 1 (true) or 0 (false) if <time1> is greater than <time2>.

lteTime <time1>[unit] <time2>[unit]

Returns a 1 (true) or 0 (false) if <time1> is less than or equal to <time2>.

gteTime <time1>[unit] <time2>[unit]

Returns a 1 (true) or 0 (false) if <time1> is greater than or equal to <time2>.

addTime <time1>[unit] <time2>[unit]

Returns the sum of adding <time1> to <time2>

subTime <time1>[unit] <time2>[unit]

Returns the value of subtracting <time2> from <time1>

mulTime <time1>[unit] <integer>

Returns the value of multiplying <time1> by an <integer>

divTime <time1>[unit] <time2>[unit]

Returns an integer, that is the value of dividing <time1> by <time2>. Specifying 0 for <time2> results in an error.

intToTime <high_32bit_int> <low_32bit_int>

Returns a 64-bit time value based on two 32-bit parts of a 64-bit integer. This command is useful when you've performed an integer calculation that results in a 64-bit value and need to convert it to a time unit.

scaleTime <time1>[unit] <scale_factor>

Returns a time value scaled by a real number and truncated to the current time resolution.

RealToTime <real>

Returns a time value equivalent to the specified real number and truncated to the current time resolution.

validTime <time>

Returns a 1 (true) or 0 (false) if the given string is a valid time for use with any of these Time calculations.

formatTime {+ | -} **commas** | {+ | -}**nofunit** | {+ | -}**bestunits**

Sets display properties for time values.

Arguments

- <time1>[unit] ...
 - <time> — Specified as an integer or decimal number. Current simulation units are the default unless specifying <unit>.
 - <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid VHDL time units are:
 - fs — femtosecond (10^{-15} seconds)
 - ps — picosecond (10^{-12} seconds)
 - ns — nanosecond (10^{-9} seconds)
 - us — microsecond (10^{-6} seconds)
 - ms — millisecond (10^{-3} seconds)
 - sec — second
 - min — minute (60 seconds)
 - hr — hour (3600 seconds)

Note that if you put a space between the values, you must enclose the argument in braces ({ }) or quotation marks (" ").
- <high_32bit_int> | <low_32bit_int>
 - <high_32bit_int> — The "high" part of the 64-bit integer.
 - <low_32bit_int> — The "low" part of the 64-bit integer.
- <scale_factor> — a real number to be used as scaling factor. Common values can include:
 - 0.25, 0.5, 1.5, 2, 10, 100

- {+ | -} commas — controls whether commas are displayed in time values.
 - +commas — time values include commas
 - commas — time values do not include commas
- {+ | -}nodefunit — controls whether time values display time units
 - +nodefunit — time values do not include time units and will be in current time resolution
 - nodefunit — time values may include time units
- {+ | -}bestunits — controls whether time values display the largest possible time unit, for example 8 us instead of 8,000 ns.
 - +bestunits — time values display the largest possible time unit
 - bestunits — time values display the default time unit

Examples

>ltTime 100ns 1ms

Returns:

1

>addTime {1545 ns} {455 ns}

Returns:

2 us

>gteTime "1000 ns" "1 us"

Returns:

1

>divTime 1us 10ns

Returns:

100

>formatTime +bestunit

Returns:

-commas -nodefunit +bestunits

>scaleTime 3ms 1000

Returns:

3 sec

>RealToTime 1.345e04

Returns:

13450 ns

toggle add

The **toggle add** command enables collection of toggle statistics for the specified nodes.

The allowed nodes are Verilog nets and registers and VHDL signals of type bit, bit_vector, std_ulogic, std_logic, and std_logic_vector. Also, VHDL Boolean and Integer types (including subranges) and other user-defined Enum types, as well as SystemVerilog real types are supported for use. All other types are silently ignored.

You can also collect and view toggle statistics in the ModelSim GUI. Refer to “[Coverage](#)” for details.

Syntax

```
toggle add [-exclude {<list>}] [-countlimit <int>]
           [-extendedtogglemode 1|2|3] [-full] [-in] [-inout] [-internal] [-out] [-ports] [-r]
           [-unique] [-widthlimit] <node_name>
```

Returns

Command result	Return value
no signals are added and no signals are found to be already in the toggle set	Nothing added.
no signals are added and some signals are found to be already in the toggle set	0
some signals are added	the number of bits added

Arguments

- **-exclude {<list>}**
Excludes specified bits of a bus from toggle computations and reports. Can also be used to exclude specific VHDL or SystemVerilog enums or ranges of enums from toggle coverage and reporting. Optional.

<list> is a space-separated list of integers or ranges, where a range is two integers separated by either ":" or "-". The range must be in the same ascending or descending order as the signal declaration. If a second **toggle add -exclude** command is issued on the same signal, the latest command will override the earlier one.
- **-countlimit <int>**
Limits the toggle coverage count for a toggle node. Optional. Overrides the global default value of <int> (which is '1'), set by the [ToggleCountLimit](#) *modelsim.ini* variable.
- **-extendedtogglemode 1|2|3**
Sets the desired level for extended toggles. Coverage calculation criteria for these three levels of support are:

1 — 0L->1H & 1H->0L & any one 'Z' transition (to/from 'Z')

2 — 0L->1H & 1H->0L & one transition to 'Z' & one transition from 'Z'

3 — 0L->1H & 1H->0L & all 'Z' transitions

- -full

Enables extended mode toggle coverage, which tracks the following six transitions:

- 1 or H --> 0 or L
- 0 or L --> 1 or H
- Z --> 1 or H
- Z --> 0 or L
- 1 or H --> Z
- 0 or L --> Z

Optional. By default only the first two transitions – to and from 0 and to and from 1 are counted. If you do a **toggle add** command on a group of signals and then try to convert to extended toggle coverage mode (all six transitions) by doing **toggle add -full** on the same signals, nothing will change. The only way to change the internal toggle triggers from default to extended toggle coverage is to restart vsim and start with the correct command.

- -in

Enables toggle statistics collection on nodes of mode IN. Optional.

- -inout

Enables toggle statistics collection on nodes of mode INOUT. Optional.

- -internal

Enables toggle statistics collection on internal (non-port) objects. Optional.

- -out

Enables toggle statistics collection on nodes of mode OUT. Optional.

- -ports

Enables toggle statistics collection on nodes of modes IN, OUT, or INOUT. Optional.

- -r

Specifies that toggle statistics collection is enabled recursively into subregions. Optional. If omitted, toggle statistic collection is limited to the current region.

- -unique

Reports an attempt to add a signal that is an alias to a signal already added. The alias will not be added. Optional.

- **-widthlimit**
Limits the maximum width of signals included in toggle coverage for the specified node. Optional. Overrides the global default limit (128) set by the [ToggleCountLimit](#) *modelsim.ini* variable.
- **<node_name>**
Enables toggle statistics collection for the named node(s). Required. Multiple names and wildcards are accepted.

Examples

- Enable toggle statistics collection for signal */dut/data/a*.

```
toggle add /dut/data/a
```

- Enable toggle statistics collection for bit 6 of bus */dut/data_in*. The curly braces must be added in order to escape the square brackets ('[]')

```
toggle add {/dut/data_in[5]}
```

See also

[“Toggle Coverage”](#), ["Toggle Exclusion Management"](#), [toggle disable](#), [toggle enable](#), [toggle report](#), [toggle reset](#)

toggle disable

The **toggle disable** command disables toggle coverage statistics collection on the specified nodes. The command provides a method of implementing coverage exclusions for toggle coverage. An equivalent command for excluding toggles from coverage is “coverage exclude -togglenode”.

For information on disabling individual bits, see [Exclude Nodes from Toggle Coverage](#) in the User’s Manual.

Syntax

```
toggle disable [-all] | [-in] [-out] [-inout] [-internal] [-ports] [-r] <node_name>
```

Arguments

- -all
Disables toggle statistics collection for all nodes that have toggle checking enabled. Optional. Must be used alone without other arguments.
- -in
Disables toggle statistics collection on nodes of mode IN. Optional.
- -out
Disables toggle statistics collection on nodes of mode OUT. Optional.
- -inout
Disables toggle statistics collection on nodes of mode INOUT. Optional.
- -internal
Disables toggle statistics collection on internal (non-port) objects. Optional.
- -ports
Disables toggle statistics collection on nodes of modes IN, OUT, or INOUT. Optional.
- -r
Specifies that toggle statistics collection is disabled recursively into subregions. Optional. If omitted, the disable is limited to the current region.
- <node_name>
Disables toggle statistics collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

“Toggle Coverage”, [toggle add](#), [toggle enable](#), [toggle report](#), [toggle reset](#), [coverage exclude](#)

toggle enable

The **toggle enable** command re-enables toggle statistics collection on nodes whose toggle coverage had previously been disabled.

Syntax

```
toggle enable [-all] | [-in] [-out] [-inout] [-internal] [-ports] [-r] <node_name>
```

Arguments

- -all
Enables toggle statistics collection for all nodes that have toggle checking disabled. Optional. Must be used alone without other arguments.
- -in
Enables toggle statistics collection on disabled nodes of mode IN. Optional.
- -out
Enables toggle statistics collection on disabled nodes of mode OUT. Optional.
- -inout
Enables toggle statistics collection on disabled nodes of mode INOUT. Optional.
- -internal
Enables toggle statistics collection on disabled internal (non-port) objects. Optional.
- -ports
Enables toggle statistics collection on disabled nodes of modes IN, OUT, or INOUT. Optional.
- -r
Specifies that toggle statistics collection is enabled recursively into subregions. Optional. If omitted, the enable is limited to the current region.
- <node_name>
Enables toggle statistics collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

[“Toggle Coverage”](#), [“Toggle Exclusion Management”](#), [toggle add](#), [toggle disable](#), [toggle report](#), [toggle reset](#), [coverage exclude](#)

toggle report

The **toggle report** command displays a list of all unique nodes that have not transitioned to both 0 and 1 at least once, and the counts for how many times each node toggled for each state transition type.

Also displayed is a summary of the number of nodes checked, the number that toggled, the number that didn't toggle, and a percentage that toggled.

You can also collect and view toggle statistics in the ModelSim GUI. Refer to “[Coverage](#)” for details.

The **toggle report** command is intended to be used as follows:

1. Enable statistics collection with the [toggle add](#) command.
2. Run the simulation with the [run](#) command.
3. Produce the report with the **toggle report** command.

Note



If you want to ensure that you are reporting all signals in the design, use the **-nocollapse** argument to **vsim** when you load your design. Without this argument, the simulator collapses certain ports that are connected to the same signal in order to improve performance, and those collapsed signals will not appear in the report. The **-nocollapse** argument degrades simulator performance, so it should be used only when it is absolutely necessary to see all signals in a toggle report.

Ordering of toggle nodes

The ordering of nodes in the report may vary depending on how you specify the signal list. If you use a wildcard in the signal argument (e.g., `toggle report -all -r /*`), the nodes are listed in the order signals are found when searching down the context tree using the wildcard. Multiple elements of the same net will be listed multiple times. If you do not use a wildcard (e.g., `toggle report -all -r /*`), the nodes are listed in the order in which they were originally added to toggle coverage, and elements are not duplicated.

Syntax

```
toggle report [-all] [-duplicates] [-file <filename>]
             [-select {inputs | outputs | inout | ports | internals}]
             [-instance <path> [-recursive]] [-onexit] [<signal>...] [-showambiguity] [-summary] [-top]
             [-verbose]
```

Arguments

- **-all**
Lists all nodes, both toggled and untoggled. Optional.
- **-duplicates**
Reports all the nodes, including alias nodes, using their local names. Optional.

- **-file <filename>**
Specifies a file to which to write the report. By default the report is displayed in the Transcript window. Optional.
- **-select {inputs | outputs | inout | ports | internals}**
Reports on input, output, inout, all ports, or internal signals. Optional.
- **-instance <path> [-recursive]**
Reports on toggles for a specified instance, and optionally on toggles under the specified instance path. Optional.

The optional **-recursive** argument specifies that toggle statistics reporting is enabled recursively into subregions. If omitted, toggle statistic reporting is limited to the current region.
- **-onexit**
Causes ModelSim to report toggle data automatically when the simulator exits. Optional.
- **<signal>...**
Specifies the name of a signal whose toggle statistics are to be displayed. Multiple signal names, separated by spaces, may be specified. Wildcards may be used.
- **-showambiguity**
When used, toggle report displays both minimum and maximum counts for any conflicting toggle data in a UCDB that results from a combined merge ([vcover merge -combine](#)).
- **-summary**
Selects only the summary portion of the report. Optional.
- **-top**
For signals that were added to toggle coverage using **vcom** or **vlog -cover t**, **-top** uses the name of the top-most element of multiple-segment (collapsed) nets. Optional. By default the name of the wildcard-matching segment will be used.
- **-unique**
This option is obsolete with version 6.3. By default, toggles are always unique.
- **-verbose**
Specifies that the toggle report includes all values taken on by integer variables. Optional.

See also

“[Toggle Coverage](#)”, [toggle add](#), [toggle disable](#), [toggle enable](#), [toggle reset](#)

toggle reset

The **toggle reset** command resets the toggle counts to zero for the specified nodes.

Syntax

```
toggle reset [-all] | [-in] [-out] [-inout] [-internal] [-ports] [-r] <node_name>
```

Arguments

- -all
Resets toggle statistics collection for all nodes that have toggle checking enabled. Optional. Must be used alone without other arguments.
- -in
Resets toggle statistics collection on nodes of mode IN. Optional.
- -out
Resets toggle statistics collection on nodes of mode OUT. Optional.
- -inout
Resets toggle statistics collection on nodes of mode INOUT. Optional.
- -internal
Resets toggle statistics collection on internal (non-port) objects. Optional.
- -ports
Resets toggle statistics collection on nodes of modes IN, OUT, or INOUT. Optional.
- -r
Specifies that toggle statistics collection is reset recursively into subregions. Optional. If omitted, the reset is limited to the current region.
- <node_name>
Resets toggle statistics collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

“Toggle Coverage”, [toggle add](#), [toggle disable](#), [toggle enable](#), [toggle report](#)

tr color

This command changes the color scheme of individual transactions and entire streams, either in a specific wave window or for all wave windows. It is the command equivalent of the Colors tab in the Transaction-Stream Properties dialog.

<color> in the arguments specifies the color to use. Either a standard X Window color name or an RGB value (e.g., #357f77) is accepted; multi-word names (“light blue”) must be enclosed in quotes.

If no arguments are specified, this command returns the value of each configuration item in a Tcl list.

Unique abbreviations are accepted for all arguments.

Syntax

```
tr color {-stream <stream> [<stream>] ... | -transaction <uid> [<uid>] ...} [-attrbg <color>]
[-attrtext <color>] [-border <color>] [-color <color>] [-default] [-get] [-inactive <color>]
[-namebg <color>] [-nametext <color>] [-win <wave>]
```

Arguments

- **-stream** <stream> [<stream>] ...
(required unless specifying **-transaction**.) If present, all objects specified in the tr color command are transaction streams. <stream> is the path to a transaction stream. Multiple streams are allowed, and the <stream> need not immediately follow the -stream argument. No wildcards are allowed.
- **-transaction** <uid> [<uid>] ...
(required unless specifying **-stream**.) Specifies the Unique IDs (UID) of the transactions to configure. The UID consists of dataset name and the 64-bit serial number assigned during simulation, which can be determined using the “tr uid” command.

<uid> can be specified either with full UID or just the serial number. If only the serial number is present, the current dataset as returned by the “env” command is assumed. If the full UID is used, it must be surrounded by curly braces ({}).

Multiple ID specifications are allowed, and the <uid> need not immediately follow the -transaction argument. No wildcards are allowed.
- **-attrbg** <color>
(optional) Select the color to use as the background for all attributes. See <color>.
- **-attrtext** <color>
(optional) Select the color to use for attribute text.
- **-border** <color>
(optional) Select the border color.

- `-color <color>`
(optional) Select the background color for the transaction. All other colors are chosen automatically.
- `<color>`
Specifies the color to use. Either a standard X Window color name or an RGB value (e.g., #357f77) is accepted; multi-word names (“light blue”) in quotes.
- `-default`
(optional) Removes any color overrides on the specified streams or transactions. If present, this option takes precedence over any other option that sets color.
- `-get`
(optional) Indicates that the command should return a list of the color schemes for each transaction or stream. If colors are changed by the command, this argument returns the resulting color scheme. Each scheme is itself a Tcl list with the colors listed in the following order: inactive line, border line, name background, name text, attribute background, attribute text.
- `-inactive <color>`
(optional) Select the inactive-line color.
- `-namebg <color>`
(optional) Select the color to use as the background for the transaction’s name.
- `-nametext <color>`
(optional) Select the color for the transaction name.
- `-win <wave>`
(optional) Specifies the wave window for which the changes should apply. `<wave>` is the Tk name (not the title) for the wave window. Any color changes to specific transactions take precedence over color changes to the streams carrying those transactions. You can change the scheme for the associated streams and not change those transactions. To remove color changes on specific transactions, use the **-default** option. The selected transactions would then reflect the color scheme of the stream.

Examples

- Set colors for the name and background for a specified transaction stream:

```
tr color -stream -namebg "light blue" -nametext black /path/tr03
```

- Set the color of the border for a specified transaction:

```
tr color -transaction -border #357f77 {sim 10023}
```

Related Topics

- [Recording and Viewing Transactions](#)
- [tr uid](#)
- [tr order](#)

tr order

This command controls which attributes are visible and the order in which they appear. It applies to entire streams only, either in a specific wave window or for all wave windows. It is the command equivalent of the Order tab in the Transaction-Stream Properties dialog.

This command functions to either:

- specify which attributes are visible and the order of those attributes (using **-attributes** and **-default**)
- display the attribute order and visibility settings (using **-hidden** and **-visible**).

Because two streams may have different attributes or the same attributes in different order, this command resolves the differences when setting the attribute order and visibility. When you set the order with **-attributes**, only attributes applying to a specific stream are visible. All other attributes for that stream are hidden. Names not matching actual attributes are ignored for that stream.

When restoring the original order with **-default**, each stream returns to its original order and visibility which may be different from that of another stream in the command line.

Unique abbreviations are accepted for all arguments.

Syntax

```
tr order [[-attributes <attrs>] [-default] [-win <wave>] <stream> ...] |  
        [-hidden] [-visible] [-win <wave>] {<stream> ...}]
```

Arguments

- **-attributes <attrs>**
(optional) Specifies that the attributes for the specified stream(s) should be visible. All other attributes are hidden. The order of the list determines the order of the attributes listed. <attrs> is a Tcl list of attribute names. Use { } to specify that there should be no visible attributes.
- **-default**
(optional) Removes any visibility and order overrides on the specified streams or transactions. If present, this option takes precedence over the **-attributes** option.
- **-hidden**
(optional) Return a list of the hidden attributes for each stream or transaction specified. If **-visible** is set, the hidden attributes are in a list following the visible attributes.
- **-visible**
(optional) Return a list of the visible attributes for each stream or transaction specified. If **-hidden** is set, the visible attributes are in a list preceding the hidden attributes.

tr order

- `-win <wave>`
(optional) If present, this option specifies the wave window for which the changes should apply. `<wave>` is the Tk name (not the title) for the wave window for which the changes apply.
- `<stream> ...`
(required) The path to a transaction stream. Multiple streams are allowed as a space separated list. No wildcards are allowed. The `tr order` command requires either a stream or unique ID.

Examples

- Set the order in which attributes appear in the wave window for a specified transaction stream:

```
tr order -attr attr1 attr2 attr3 /path/tr03
```

- Returns the attribute order for the `top/stream` stream to the default order:

```
tr order -default /top/stream
```

- Sets the visibility for attributes of the transaction stream `/top/stream2`:

```
tr order -visible attr1 -hidden attr2 top/stream2
```

Displays a Tcl list of visible attributes, followed by the hidden attributes.

Related Topics

- [Recording and Viewing Transactions](#)
- [tr color](#)
- [tr uid](#)

tr uid

This command returns a list of unique transaction IDs for the specified time span on the specified streams. A transaction UID is the logical name of its dataset and its a 64-bit serial number created during simulation.

Usage: you can pass the returned UIDs to the **tr color** command to specify a particular transaction.

The returned UIDs represent transactions that are ACTIVE during the time span. If a transaction starts anywhere in the time span, at the start of the span or even at the end of the span, it is considered active. A transaction that ends at the start time is not active.

The optional arguments in this command apply either to a:

- listing of transactions occurring over a large range of time (using **-end** and **-start**)
- listing of transactions that are active at one specific time (using **-time**)

Unique abbreviations are accepted for all arguments.

Syntax

```
tr uid [-time <time> <stream> ...] | [-start <time> -end <time> <stream> ... ]
```

Arguments

- **-end**
(required in conjunction with the **-start** option) Specifies the start of the span of time from which UIDs should be obtained.
- **<time>**
(required) Specifies time, or time and delta (must be a positive integer).
- **-start**
(required in conjunction with the **-end** option) Specifies the start of the span of time from which UIDs should be obtained.
- **-time**
(required unless **-end** and **-start** are specified.) Specifies a point in time for the command.
- **<stream> ...**
(required) The path to a transaction stream. May specify more than one stream. No wildcards are allowed.

Examples

- List all transaction UIDs for a specified transaction stream:

```
tr uid /path/tr03
```

- List the transaction UID for a specified transaction stream at a particular time:

tr uid -time 20ns {sim 209456}

Related Topics

- [Recording and Viewing Transactions](#)
- [tr color](#)
- [tr order](#)

transcribe

This command displays a command in the Transcript window, and then executes the command.

The transcribe command is normally used to direct commands to the Transcript window from an external event such as a menu pick or button selection. The [add button](#) and [add_menuitem](#) commands can use **transcribe**.

Returns nothing.

Syntax

```
transcribe <command>
```

Arguments

- <command>
(required) Specifies the command to execute.

Examples

- Create a button labeled "pwd" that invokes **transcribe** with the **pwd** Tcl command, and echoes the command and its results to the Transcript window. The button remains active during a run.

```
add button pwd {transcribe pwd} NoDisable
```

Related Topics

- [add button](#)
- [add_menuitem](#)

transcript

This command controls echoing of commands executed in a macro file.

If no option is specified, the current setting is reported.

Syntax

```
transcript [on | off | -q | quietly]
```

Arguments

- on
(optional) Specifies that commands in a macro file will be echoed to the Transcript window as they are executed.
- off
(optional) Specifies that commands in a macro file will not be echoed to the Transcript window as they are executed.
The [transcribe](#) command can be used to force a command to be echoed.
- -q
(optional) Returns "0" if transcribing is turned off or "1" if transcribing is turned on. Useful in a Tcl conditional expression.
- quietly
(optional) Turns off the transcript echo for all commands. To turn off echoing for individual commands see the [quietly](#) command.

Examples

- Commands within a macro file will be echoed to the Transcript window as they are executed.

```
transcript on
```

- If issued immediately after the previous example, the message:

```
transcript
```

Returns

```
Macro transcribing is turned ON.
```

Related Topics

- [Transcript Window](#)
- [echo](#)
- [transcribe](#)
- [.main clear](#)

transcript file

This command sets or queries the current PrefMain(file) Tcl preference variable. You can use this command to clear a transcript in batch mode or to limit the size of a transcript file. It offers an alternative to setting the PrefMain(file) Tcl preference variable through the GUI.

Syntax

```
transcript file [<filename>]
```

Arguments

- <filename>

(optional) Specifies a name for the transcript file. Wildcard characters are allowed, and “stdout” or “stderr” are valid file names. If you specify a new file, the existing transcript file is closed and a new transcript file opened. If you specify an empty string (“”), the existing file is closed and no new file is opened. If you don’t specify this argument, the current filename is returned.

Note



You can prevent overwriting older transcript files by including a pound sign (#) in <filename> when <filename> is a repeated string. The simulator replaces the pound character (#) with the next available sequence number when saving a new transcript file.

Examples

- Close the current transcript file and stops writing data to the file. This is a method for reducing the size of your transcript.

```
transcript file ""
```

- Close the current transcript file named *trans1.txt* and open a new transcript file, incrementing the file name by 1.

```
transcript file trans#.txt
```

Closes *trans1.txt* and opens *trans2.txt*.

- This series of commands results in the transcript containing only data from the second millisecond of the simulation. The first **transcript file** command closes the transcript so no data is being written to it. The second **transcript file** command opens a new transcript and records data from 1 ms to 2 ms.

```
transcript file ""  
run 1 ms  
transcript file transcript  
run 1 ms
```

Related Topics

- ["Creating a Transcript File"](#)
- ["Setting Preference Variables from the GUI"](#)
- [Transcript Window](#)
- [.main clear](#)
- [transcript path](#)
- [transcript sizelimit](#)

transcript path

This command returns the full pathname to the current transcript file.

Syntax

transcript path

Arguments

- None

Related Topics

- ["Creating a Transcript File"](#)
- ["Setting Preference Variables from the GUI"](#)
- ["Transcript Window"](#)
- [transcript file](#)

transcript sizelimit

This command sets or queries the current preference value for the transcript fileSizeLimit value.
If the size limit is reached, the transcript file is saved and simulation continues.

Syntax

```
transcript sizelimit [<size>]
```

Arguments

- <size>
(optional) Specifies the size, in KB, of the transcript file where the default is 0 or unlimited. The actual file size may be larger by as much as one buffer size (usually about 4k), depending on the operating system default buffer size and the size of the messages written to the transcript.

Note



You can set the size of the transcript file with the \$PrefMain (fileSizeLimit) Tcl variable in the Preferences dialog. Refer to "[Setting Preference Variables from the GUI](#)" for more information.

Related Topics

- "[Creating a Transcript File](#)"
- "[Setting Preference Variables from the GUI](#)"
- "[Transcript Window](#)"
- [transcript file](#)

tssi2mti

This command is used to convert a vector file in TSSI Format into a sequence of force and run commands.

The stimulus is written to the standard output.

The source code for **tssi2mti** is provided in the file *tssi2mti.c* in the *examples* directory.

Syntax

```
tssi2mti <signal_definition_file> [<sef_vector_file>]
```

Arguments

- <signal_definition_file>
(required) Specifies the name of the TSSI signal definition file describing the format and content of the vectors.
- <sef_vector_file>
(optional) Specifies the name of the file containing vectors to be converted. If none is specified, standard input is used.

Examples

- The command will produce a do file named *trigger.do* from the signal definition file *trigger.def* and the vector file *trigger.sef*.

```
tssi2mti trigger.def trigger.sef > trigger.do
```

- This example is the same as the previous one, but uses the standard input instead.

```
tssi2mti trigger.def < trigger.sef > trigger.do
```

Related Topics

- [force](#)
- [run](#)
- [write tssi](#)

typespec

This command queries class names and class relationships of SystemVerilog classes.

Syntax

```
typespec <pattern> [-isa <value>] [-class <value> | ancestry [-indent <value>]]  
    [-exact | -regexp]
```

Arguments

- <pattern>
(optional) The pattern you are querying for.
- -isa <value>
(optional) Returns classes derived from the given <value>.
- -class <value>
(optional) Returns specialized classes of the given <value>, where <value> represents a parameterized class and the results are specific instances of the class with given parameter values.
- -ancestry <value>
(optional) Returns a list of base classes for the given <value>.
- -indent <value>
(optional) Prefixes each level of the class inheritance hierarchy with <value>.
- -exact
(optional) Returns results that match the given <pattern> exactly.
- -regexp
(optional) Specifies that all <pattern> arguments should be treated as regular expressions.

ui_VVMode

This command specifies behavior when encountering UI registration calls used by verification packages, such as AVM or OVM. Returns the current setting when specifies without an argument.

Syntax

```
ui_VVMode [full | logclass | logobj | nolog | off ]
```

Arguments

- full
(optional) Enables the context registration of the UI registration call and automatically logs both the class type and the registered object to the WLF file.
- logclass
(optional) Enables the context registration of the UI registration call and automatically logs the class type of the registered object to the WLF file.
- logobj
(optional) Enables the context registration of the UI registration call and automatically logs the registered object to the WLF file
- nolog
(optional) Enables the context registration of the UI registration call, but does not automatically log the registration to the WLF file. (default)
- off
(optional) Disables context registration and automatic logging when encountering UI registration calls.

Description

UI registration calls, Verilog system tasks specific to this product, are typically included in verification packages such as AVM and OVM so that key information about the packages is available when debugging the simulation. The UI registration calls include:

- \$ui_VVInstallInst() — Defines a region in the context tree, which will appear in the Structure window.
- \$ui_VVInstallObj() — Adds an object to the defined parent, which will appear in the Objects window when the parent instance is selected in the Structure window.
- \$ui_VVInstallPort() — Adds a port that is an object that connects to another component, which will appear in the Objects window when the parent instance is selected in the Structure window.
- \$ui_VVSetFilter() — Specifies which class properties should not be shown in the GUI.

- `$ui_VVSetAllow()` — Specifies which class properties should be retained that were filtered out with `$ui_VVSetFilter`.

unsetenv

This command deletes an environment variable. The deletion is not permanent – it is valid only for the current ModelSim session.

Syntax

```
unsetenv <varname>
```

Arguments

- <varname>
(required) The name of the environment variable you wish to delete.

Related Topics

- [setenv](#)
- [printenv](#)

up

This command searches for object transitions or values in the specified List window.

It executes the search on objects currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which an object takes on a particular value, or an expression of multiple objects evaluates to true. Refer to the [down](#) command for related functionality.

Prerequisites

The procedure for using **up** entails three steps:

1. Click on the desired waveform.
2. Click on the desired starting location.
3. Issue the **up** command. (The [seetime](#) command can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

Syntax

```
up [-expr {<expression>}] [-falling] [-noglitch] [-rising] [-value <sig_value>]
  [-window <wname>] [<n>]
```

Arguments

- -expr {<expression>}

(optional) Searches the waveform display for an expression. When the search evaluates to a boolean true, the active cursor moves to the found location. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced Wave window. A signal may be specified either by its full path or by the shortcut label displayed in the Wave window.

{<expression>} — Any valid expression. See [GUI_expression_format](#) for the format of the expression. The expression must be placed within curly braces.
- -falling

(optional) Searches for a falling edge on the specified object if that object is a scalar. If it is not a scalar, the option will be ignored. Optional.
- -noglitch

Specifies that delta-width glitches are to be ignored.
- -rising

(optional) Searches for a rising edge on the specified object if that object is a scalar. If it is not a scalar, the option will be ignored.

- `-value <sig_value>`
(optional) Specifies a value of the signal to match. Only one signal may be selected, but that signal may be an array.
`<sig_value>` — Must be specified in the same radix that the selected waveform is displayed. Case is ignored, but otherwise the value must be an exact string match -- don't-care bits are not yet implemented.
- `-window <wname>`
(optional) Specifies an instance of the Wave window that is not the default. Otherwise, the default Wave window is used. Use the [view](#) command to change the default window.
`<wname>` — The name of a Wave window other than the current default window.
- `<n>`
(optional) Specifies to find the nth match. If less than n are found, the number found is returned with a warning message, and the marker is positioned at the last match.

Examples

- Find the last time at which the selected vector transitions to FF23, ignoring glitches.
`up -noglitch -value FF23`
- Go to the previous transition on the selected object.
`up`

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the [GUI_expression_format](#).

- Search up for an expression that evaluates to a boolean 1 when signal *clk* just changed from low to high and signal *mystate* is the enumeration reading and signal */top/u3/addr* is equal to the specified 32-bit hex constant.
`up -expr {clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)}`
- Search up for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal */top/u3/addr* equals hex ac.
`up -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}`
- Search up for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, clock just changed from low to high, and signal *mode* is enumeration writing.
`up -expr {{{(NOW > 23 us) && (NOW < 54 us)} && clk'rising && (mode == writing)}}`

Related Topics

- [GUI_expression_format](#)
- [down](#)
- [seetime](#)
- [view](#)

vcd add

This command adds the specified objects to a VCD file.

The allowed objects are Verilog nets and variables and VHDL signals of type `bit`, `bit_vector`, `std_logic`, and `std_logic_vector` (other types are silently ignored). The command works with mixed HDL designs.

All **vcd add** commands must be executed at the same simulation time. The specified objects are added to the VCD header and their subsequent value changes are recorded in the specified VCD file. By default all port driver changes and internal variable changes are captured in the file. You can filter the output using arguments detailed below.

Related Verilog tasks: `$dumpvars`, `$fdumpvars`

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
vcd add [-dumpports] [-file <filename>] [[-in] [-out] [-inout] | [-ports]] [-internal]
        [-r | -r -optcells] <object_name> ...
```

Arguments

- **-dumpports**
(optional) Specifies port driver changes to be added to an extended VCD file. When the **vcd dumpports** command cannot specify all port driver changes that will appear within the VCD file, multiple **vcd add -dumpports** commands can be used to specify additional port driver changes.
- **-file <filename>**
(optional) Specifies the name of the VCD file. This option should be used only when you have created multiple VCD files using the **vcd files** command.
`<filename>` — A *.vcd* file.
- **-in**
(optional) Includes only port driver changes from ports of mode IN.
- **-out**
(optional) Includes only port driver changes from ports of mode OUT.
- **-inout**
(optional) Includes only port driver changes from ports of mode INOUT.
- **-ports**
(optional) Includes only port driver changes. Excludes internal variable or signal changes.
- **-internal**
(optional) Includes only internal variable or signal changes. Excludes port driver changes.

- `-r | -r -optcells`
(optional) Specifies that signal and port selection occurs recursively into subregions. If omitted, included signals and ports are limited to the current region. When `-r` is used with `-optcells` it allows Verilog optimized cell ports to be visible when using wildcards. By default Verilog optimized cell ports are not selected even if they match the specified wildcard pattern.
- `<object_name> ...`
(required) Specifies the Verilog or VHDL object or objects to add to the VCD file. Multiple objects may be specified by separating names with spaces. Wildcards are accepted. Must be specified as the final argument to the **vcd add** command.

Related Topics

- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd checkpoint

This command dumps the current values of all VCD variables to the specified VCD file. While simulating, only value changes are dumped.

Related Verilog tasks: \$dumpall, \$fdumpall

Syntax

```
vcd checkpoint [<filename>]
```

Arguments

- <filename>
(optional) Specifies the name of the VCD file. If omitted the command is executed on the file designated by the [vcd file](#) command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- [vcd add](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd comment

This command inserts the specified comment in the specified VCD file.

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
vcd comment <comment string> [<filename>]
```

Arguments

- <comment string>
(required) Comment to be included in the VCD file. Must be enclosed by double quotation marks or curly braces. Must be specified as the first argument to the **vcd comment** command.
- <filename>
(optional) Specifies the name of the VCD file. If omitted the command is executed on the file designated by the [vcd file](#) command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpports

This command creates a VCD file that includes port driver data.

By default all port driver changes are captured in the file. You can filter the output using arguments detailed below. Related Verilog task: \$dumpports

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
vcd dumpports [-compress] [-direction] [-file <filename>] [-force_direction] [-in] [-out] [-inout] [-no_strength_range] [-unique] [-vcdstim] <object_name> ...
```

Arguments

- **-compress**
(optional) Produces a compressed VCD file. ModelSim uses the gzip compression algorithm. It is not necessary to specify **-compress** if you specify a *.gz* extension with the **-file <filename>** argument
- **-direction**
(optional) Includes driver direction data in the VCD file.
- **-file <filename>**
(optional) Creates a VCD file. Defaults to the current working directory and the filename *dumpports.vcd*. Multiple filenames can be opened during a single simulation.
<filename> — Specifies a filename. When specified with a *.gz* extension, the file is compressed.
- **-force_direction**
(optional) Causes **vcd dumpports** to use the specified port direction (instead of driver location) to determine whether the value being dumped is input or output. This argument overrides the default use of the location of drivers on the net to determine port direction (this is because Verilog port direction is not enforced by the language or by ModelSim).
- **-in**
(optional) Includes ports of mode IN.
- **-out**
(optional) Includes ports of mode OUT.
- **-inout**
(optional) Includes ports of mode INOUT.

- **-no_strength_range**
(optional) Ignores strength ranges when resolving driver values. This argument is an extension to the IEEE 1364 specification. Refer to [Resolving Values](#) for additional information.
- **-unique**
(optional) Generates unique VCD variable names for ports even if those ports are connected to the same collapsed net.
- **-vcdstim**
(optional) Ensures that port name order in the VCD file matches the declaration order in the instance module or entity declaration. Refer to [Port Order Issues](#) for further information.
- **<object_name> ...**
(required) Specifies one or more Verilog, VHDL, or SystemC objects to add to the VCD file. You can specify multiple objects by separating names with spaces. Wildcards are accepted. Must be specified as the final argument to the **vcd dumpports** command.

Examples

- Create a VCD file named *counter.vcd* of all IN ports in the region */test_design/dut/*.

```
vcd dumpports -in -file counter.vcd /test_design/dut/*
```

- These two commands resimulate a design from a VCD file. Refer to [Simulating with Input Values from a VCD File](#) for further details.

```
vcd dumpports -file addern.vcd /testbench/uut/*  
vsim -vcdstim addern.vcd addern -gn=8 -do "add wave /*; run 1000"
```

- This series of commands creates VCD files for the instances *proc* and *cache* and then re-simulates the design using the VCD files in place of the instance source files. Refer to [Replacing Instances with Output Values from a VCD File](#) for more information.

```
vcd dumpports -vcdstim -file proc.vcd /top/p/*  
vcd dumpports -vcdstim -file cache.vcd /top/c/*  
run 1000  
  
vsim top -vcdstim /top/p=proc.vcd -vcdstim /top/c=cache.vcd
```

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpportsall

This command creates a checkpoint in the VCD file which shows the value of all selected ports at that time in the simulation, regardless of whether the port values have changed since the last timestep.

Related Verilog task: \$dumpportsall

Syntax

```
vcd dumpportsall [<filename>]
```

Arguments

- <filename>
(optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpportsflush

This command flushes the contents of the VCD file buffer to the specified VCD file.

Related Verilog task: \$dumpportsflush

Syntax

```
vcd dumpportsflush [<filename>]
```

Arguments

- `<filename>`
(optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpportslimit

This command specifies the maximum size of the VCD file (by default, limited to available disk space). When the size of the file exceeds the limit, a comment is appended to the file and VCD dumping is disabled.

Related Verilog task: `$dumpportslimit`

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
vcd dumpportslimit <dumplimit> [<filename>]
```

Arguments

- `<dumplimit>`
(required) Specifies the maximum VCD file size in bytes. Must be specified as the first argument to the **vcd dumpportslimit** command.
- `<filename>`
(optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpportsoff

This command turns off VCD dumping and records all dumped port values as x.

Related Verilog task: \$dumpportsoff

Syntax

```
vcd dumpportsoff [<filename>]
```

Arguments

- <filename>
(optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpportson

This command turns on VCD dumping and records the current values of all selected ports. This command is typically used to resume dumping after invoking [vcd dumpportsoff](#).

Related Verilog task: \$dumpportson

Syntax

```
vcd dumpportson [<filename>]
```

Arguments

- `<filename>`
(optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd file

This command specifies the filename and state mapping for the VCD file created by a **vcd add** command. The **vcd file** command is optional. If used, it must be issued before any **vcd add** commands.

Related Verilog task: \$dumpfile

Syntax

```
vcd file [-dumpports] [-direction] [<filename>] [-map <mapping pairs>] [-no_strength_range]
        [-nomap] [-unique]
```

Arguments

- **-dumpports**
(optional) Capture detailed port driver data for Verilog ports and VHDL `std_logic` ports. This option works only on ports, and any subsequent **vcd add** command will accept only qualifying ports (silently ignoring all other specified objects).
- **-direction**
(optional) Includes driver direction data in the VCD file.
- **<filename>**
(optional) Specifies the name of the VCD file that is created where the default is *dump.vcd*.
- **-map <mapping pairs>**
(optional) Overrides the default mappings. Affects only VHDL signals of type `std_logic`.
<mapping pairs> — Specified as a list of character pairs. The first character in a pair must be one of the `std_logic` characters UX01ZWLH- and the second character is the character you wish to be recorded in the VCD file. The Tcl convention for command strings that include spaces is to enclose them in quotation marks (“ ”). For example, to map L and H to z:

```
vcd file -map "L z H z"
```
- **-no_strength_range**
(optional) Ignores strength ranges when resolving driver values. This argument is an extension to the IEEE 1364 specification. Refer to [Resolving Values](#) for additional information.
- **-nomap**
(optional) Affects only VHDL signals of type `std_logic`. It specifies that the values recorded in the VCD file shall use the `std_logic` enumeration characters of UX01ZWLH-. This option results in a non-standard VCD file because VCD values are limited to the four state character set of x01z. By default, the `std_logic` characters are mapped as follows.

VHDL	VCD	VHDL	VCD
U	x	W	x
X	x	L	0
0	0	H	1
1	1	-	x
Z	z		

- -unique

(optional) Generates unique VCD variable names for ports even if those ports are connected to the same collapsed net.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd files

This command specifies filenames and state mapping for VCD files created by the **vcd add** command. The **vcd files** command is optional. If used, it must be issued before any **vcd add** commands.

Related Verilog task: \$fdumpfile

Syntax

```
vcd files [-compress] [-direction] <filename> [-map <mapping pairs>] [-no_strength_range]
         [-nomap] [-unique]
```

Arguments

- **-compress**
(optional) Produces a compressed VCD file. ModelSim uses the gzip compression algorithm. If you specify a .gz extension on the **-file <filename>** argument, ModelSim compresses the file even if you don't use the **-compress** argument.
- **-direction**
(optional) Includes driver direction data in the VCD file.
- **<filename>**
(required) Specifies the name of a VCD file to create. Multiple files can be opened during a single simulation; however, you can create only one file at a time. If you want to create multiple files, invoke **vcd files** multiple times.
- **-map <mapping pairs>**
(optional) Overrides the default mappings. Affects only VHDL signals of type `std_logic`.
<mapping pairs> — Specified as a list of character pairs. The first character in a pair must be one of the `std_logic` characters UX01ZWLH- and the second character is the character you wish to be recorded in the VCD file. The Tcl convention for command strings that include spaces is to enclose them in quotation marks (“”). For example, to map L and H to z:

```
vcd file -map "L z H z"
```
- **-no_strength_range**
(optional) Ignores strength ranges when resolving driver values. This argument is an extension to the IEEE 1364 specification. Refer to “[Resolving Values](#)” for additional information.
- **-nomap**
(optional) Affects only VHDL signals of type `std_logic`. It specifies that the values recorded in the VCD file shall use the `std_logic` enumeration characters of UX01ZWLH-. This option

results in a non-standard VCD file because VCD values are limited to the four state character set of x01z. By default, the std_logic characters are mapped as follows.

VHDL	VCD	VHDL	VCD
U	x	W	x
X	x	L	0
0	0	H	1
1	1	-	x
Z	z		

- `-unique`
(optional) Generates unique VCD variable names for ports even if those ports are connected to the same collapsed net.

Examples

The following example shows how to "mask" outputs from a VCD file until a certain time after the start of the simulation. The example uses two **vcd files** commands and the `vcd on` and `vcd off` commands to accomplish this task.

```
vcd files in_inout.vcd
vcd files output.vcd
vcd add -in -inout -file in_inout.vcd /*
vcd add -out -file output.vcd /*
vcd off output.vcd
run 1us
vcd on output.vcd
run -all
```

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd flush

This command flushes the contents of the VCD file buffer to the specified VCD file. This command is useful if you want to create a complete VCD file without ending your current simulation.

Related Verilog tasks: \$dumpflush, \$fdumpflush

Syntax

```
vcd flush [<filename>]
```

Arguments

- <filename>
(optional) Specifies the name of the VCD file. If omitted, the command is executed on the file designated by the [vcd file](#) command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd limit

This command specifies the maximum size of a VCD file (by default, limited to available disk space).

When the size of the file exceeds the limit, a comment is appended to the file and VCD dumping is disabled.

Related Verilog tasks: \$dumplimit, \$fdumplimit

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
vcd limit <filesize> [<filename>]
```

Arguments

- <filesize>
(Required) Specifies the maximum VCD file size, in bytes. The numerical value of <filesize> can only be a whole number. Must be specified as the first argument to the **vcd limit** command.

You can specify a unit of Kb, Mb, or Gb with the numerical value (units are case insensitive). Do not insert a space between the numerical value and the unit (for example, 400Mb, not 400 Mb).
- <filename>
(Optional) Specifies the name of the VCD file. If omitted, the command is executed on the file designated by the **vcd file** command or *dump.vcd* if **vcd file** was not invoked.

Example

- Specify a maximum VCD file size of 6 gigabytes and a VCD file named my_vcd_file.vcd.

```
vcd limit 6gb my_vcd_file.vcd
```

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd off](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd off

This command turns off VCD dumping to the specified file and records all VCD variable values as x.

Related Verilog tasks: \$dumpoff, \$fdumpoff

Syntax

```
vcd off [<filename>]
```

Arguments

- <filename>
(optional) Specifies the name of the VCD file. If omitted, the command is executed on the file designated by the [vcd file](#) command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd on](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd on

This command turns on VCD dumping to the specified file and records the current values of all VCD variables.

By default, **vcd on** is automatically performed at the end of the simulation time that the **vcd add** command performed.

Related Verilog tasks: \$dumpon, \$fdumpon

Syntax

```
vcd on [<filename>]
```

Arguments

- <filename>
(optional) Specifies the name of the VCD file. If omitted the command is executed on the file designated by the **vcd file** command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd2wlf](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcd2wlf

This command is a utility that translates a VCD (Value Change Dump) file into a WLF file that you can display in ModelSim using the **vsim -view** argument. This command only works on VCD files containing positive time values.

Description

The **vcd2wlf** command functions as simple one-pass converter. If you are defining a bus in a VCD file, you must specify all bus bits before the next \$scope or \$upscope statement appears in the file. The best way to ensure that bits get converted together as a bus is to declare them on consecutive lines.

For example:

```
Line 21 : $var wire 1 $ in [2] $end
Line 22 : $var wire 1 $u in [1] $end
Line 23 : $var wire 1 # in [0] $end
```

Arguments to this command are order dependent. Please read the argument descriptions for more information.

Syntax

```
vcd2wlf [-splitio] [-splitio_in_ext <extension>] [-splitio_out_ext <extension>] [-nocase]
        <vcd filename> <wlf filename>
```

Arguments

- **-splitio**
(optional) Specifies that extended VCD port values are to be split into their corresponding input and output components by creating two signals instead of just one in the resulting *.wlf* file. By default the new input-component signal keeps the same name as the original port name while the output-component name is the original name with "**__o**" appended to it.
- **-splitio_in_ext <extension>**
(optional) Adds an extension to input-component signal names created by using **-splitio**.
<extension> — Specifies a string.
- **-splitio_out_ext <extension>**
(optional) Adds an extension to output-component signal names created by using **-splitio**.
<extension> — Specifies a string.
- **-nocase**
(optional) Converts all alphabetic identifiers to lowercase.
- **<vcd filename>**
(required) Specifies the name of the VCD file you want to translate into a WLF file. Must be specified immediately preceding the **<wlf filename>** argument to the **vcd2wlf** command.

- `<wlf filename>`
(required) Specifies the name of the output WLF file. Must be specified as the final argument to the **vcd2wlf** command.

Related Topics

- [vcd add](#)
- [vcd checkpoint](#)
- [vcd comment](#)
- [vcd dumpports](#)
- [vcd dumpportsall](#)
- [vcd dumpportsflush](#)
- [vcd dumpportslimit](#)
- [vcd dumpportsoff](#)
- [vcd dumpportson](#)
- [vcd file](#)
- [vcd files](#)
- [vcd flush](#)
- [vcd limit](#)
- [vcd off](#)
- [vcd on](#)
- [DumpportsCollapse](#)
- [Value Change Dump \(VCD\) Files](#)
- Verilog tasks are documented in the IEEE 1364 standard.

vcom

The **vcom** command compiles VHDL source code into a specified working library (or to the **work** library by default).

You can invoke **vcom** either from within ModelSim or from the command prompt of your operating system. You can invoke this command during simulation.

Compiled libraries are dependent on the major version of ModelSim. When moving between major versions, you must refresh compiled libraries using the **-refresh** argument to **vcom**. This is not required for minor versions (letter releases).

All arguments to the **vcom** command are case-sensitive. For example, **-WORK** and **-work** are not equivalent.

This command provides additional information with the **-help** switch.

Syntax

```
vcom [options] <filename> [<filename> ...]
```

[options]:

```
[-0in] [-0in_options "<options>"] [-32 | -64] [-87 | -93 | -2002 | -2008]
[+acc[=<spec>][+<entity>[(architecture)]]] [-allowProtectedBeforeBody]
[-amsstd | -noamsstd]
[-bindAtCompile] [-bindAtLoad]
[-check_synthesis] [-constimmedassert | -noconstimmedassert] [+cover[=<spec>]]
[-cover <spec>] [-coveropt <opt_level>] [-coverreportcancelled] [-coverenhanced]
[-coverexcludedefault] [-coversub | -nocoversub]
[-debugVA] [-deferSubpgmCheck | -noDeferSubpgmCheck] [-dpiforceheader]
[-error <msg_number>[,<msg_number>,...]] [-explicit] [-extendedtogglemode 1|2|3]
[-f <filename>] [-fatal <msg_number>[,<msg_number>,...]]
[-fsmimplicittrans] [-fsmmultitrans] [-fsmresettrans | -nofsmresettrans]
[-fsmsingle | -nofmsingle ] [-fsmverbose [b | t | w]] [-force_refresh <design_unit>]
[-gen_xml <design_unit> <filename>]
[-ignoredefaultbinding] [-ignorevitalerrors]
[-just abcep]
[-line <number>] [-lint]
[-maxfecrows] [-maxudprows] [-mixedsvvh [b | l | r | i]] [-modelsimini <ini_filepath>]
[-no1164] [-noaccel <package_name>] [-nocasestaticerror] [-nocheck]
[-nocoverrespecthandl] [-nodbgSYM] [-noindexcheck] [-nofsmxassign] [-nologo]
[-nonstddriverinit] [-noothersstaticerror]
[-note <msg_number> [,<msg_number>, ...]] [-novital] [-novitalcheck] [-novopt]
[-nowarn <category_number>] [-nocovershort] [-nodebug[=ports]] [-nocoverfec]
[-nocoverudp] [-nopsl]
```


[-O0 | -O1 | -O4 | -O5]
 [-pedanticerrors] [-performdefaultbinding] [+protect [=<filename>]]
 [-psltext] [-pslfile <filename>]
 [-quiet]
 [-rangecheck | -norangecheck] [-refresh]
 [-s] [-skip abcep] [-source] [-suppress <msg_number>[,<msg_number>,...]]
 [-synthprefix <prefix>]
 [-time] [-togglecountlimit <int>] [-togglewidthlimit <int>]
 [-version] [-vmake] [-vopt | -novopt]
 [-warning <msg_number>[,<msg_number>,...]] [-work <library_name>]

Arguments

- -0in
 Invokes **0-In analyze** from within ModelSim. Optional. For the invocation to work, the environment variable HOME_0IN must be set to the proper pathname.
- -0in_options "<options>"
 Passes the specified options to **0-In analyze**. Optional. The options must be enclosed in parentheses. Consult the 0-In documentation for details on the available options. For the invocation to work, the environment variable HOME_0IN must be set to the proper pathname.
- -32 | -64
 Specifies whether vcom uses the 32- or 64-bit executable, where -32 is the default.
 These options override the MTI_VCO_MODE environment variable, which applies only to executables used from the <install_dir>/bin/ directory. Therefore, these options are ignored if you run vcom from an <install_dir>/<platform>/ directory.
 You can only specify these options on the command line, therefore they are not recognized as part of a file used with the -f switch.
- -87 | -93 | -2002 | -2008
 (optional) Specifies which LRM-specific compiler to use. You can also control this behavior with the VHDL93 variable in the *modelsim.ini* file.
 - 87 — Enables support for VHDL 1076-1987.
 - 93 — Enables support for VHDL 1076-1993.
 - 2002 — Enables support for VHDL 1076-2002. (default)
 - 2008 — Enables support for VHDL 1076-2008.
- +acc[=<spec>][+<entity>[(architecture)]]
 Enable debug command access to objects indicated by <spec> when optimizing a design. Optional.

Note



Using this option may reduce simulation speed.

<spec> can be:

f —

Enable access to finite state machines

v —

Enable access to variables, constants, and aliases in processes that would otherwise be merged due to optimizations. Disables an optimization that automatically converts variables to constants.

If <spec> is omitted, access is enabled for all objects.

<entity> and (<architecture>) specify the VHDL design regions in which to allow the access. If (<architecture>) is not specified, then all architectures of a given <entity> are enabled for access. May be optionally followed by "." to indicate all children of the module.

- -allowProtectedBeforeBody

Allows a variable of a protected type to be created prior to declaring the body. Optional.

- -amsstd | -noamsstd

Specifies whether vcom adds the declaration of REAL_VECTOR to the STANDARD package. This is useful for designers using VHDL-AMS to test digital parts of their model.

-amsstd — REAL_VECTOR is included in STANDARD.

-noamsstd — REAL_VECTOR is not included in STANDARD (default).

You can also control this with the [AmsStandard](#) variable or the [MGC_AMS_HOME](#) environment variable.

- -bindAtCompile

Forces ModelSim to perform default binding at compile time rather than at load time. Optional. Refer to “[Default Binding](#)” for more information. You can change the permanent default by editing the [BindAtCompile](#) variable in the *modelsim.ini*.

- -bindAtLoad

Forces ModelSim to perform default binding at load time rather than at compile time. Optional. Default.

- -check_synthesis

Turns on limited synthesis rule compliance checking. Specifically, it checks to see that signals read by a process are in the sensitivity list. Optional. The checks understand only combinational logic, not clocked logic. Edit the [CheckSynthesis](#) variable in the *modelsim.ini* file to set a permanent default.

- -constimmedassert

Displays immediate assertions with constant expressions in the GUI, in reports, and in the UCDB. Optional. By default, immediate assertions with constant expressions are displayed in the GUI, in reports, and in the UCDB. Use this switch only if the -noconstimmedassert switch has been used previously, or if the ShowConstantImmediateAsserts variable in the vcom section of the *modelsim.ini* file is set to 0 (off).

- -noconstimmedassert

Turns off the display of immediate assertions with constant expressions in the GUI, in reports, and in the UCDB. Optional. By default, immediate assertions with constant expressions are displayed. You may also set the ShowConstantImmediateAsserts variable in the vcom section of the *modelsim.ini* file to 0 (off).

- +cover[=<spec>]

Enables various coverage statistics collection on all design units compiled in the current compiler run. Optional. Consider using the +cover argument to [vopt](#) instead, which you can use to specify the precise design units and regions to be instrumented for coverage. The +cover argument with no "=<spec>" designation is equivalent to "+cover=bcesft".

<spec> — one or more of the following characters:

b — Collect branch statistics.

c — Collect condition statistics. Collects both FEC and UDP statistics, unless -nocoverfec or -nocoverudp is specified.

e — Collect expression statistics, Collects both FEC and UDP statistics, unless -nocoverfec or -nocoverudp is specified.

s — Collect statement statistics.

t — Collect toggle statistics. Overridden if 'x' is specified elsewhere.

x — Collect extended toggle statistics (Refer to “[Toggle Coverage](#)” for details). This takes precedence, if 't' is specified elsewhere.

f — Collect Finite State Machine statistics.

See [-coveropt <opt_level>](#) argument to override the default level of optimization for coverage for a particular compilation run.

- -cover <spec>

Recommendation: Use "vopt +cover" rather than "vcom -cover", as it is more powerful and flexible, and often yields better performance. See the [vopt +cover](#) argument for more information.

Enables various coverage statistics collection. Optional.

<spec> — one or more of the following characters:

b — Collect branch statistics.

c — Collect condition statistics. Collects both FEC and UDP statistics, unless -nocoverfec or -nocoverudp is specified.

- e — Collect expression statistics, Collects both FEC and UDP statistics, unless -nocoverfec or -nocoverudp is specified.
- s — Collect statement statistics.
- t — Collect toggle statistics. Cannot be used if 'x' is specified.
- x — Collect extended toggle statistics (Refer to “[Toggle Coverage](#)” for details). Cannot be used if 't' is specified.
- f — Collect Finite State Machine statistics.

- -coverenhanced

Enables non-critical functionality which might change the appearance or content of coverage metrics. This argument only has an effect in letter releases (6.6a, 6.6b, etc.) . In major releases (6.6, 6.7, etc.), all coverage enhancements present in previous letter release streams are enabled by default, and -coverenhanced is no longer necessary to enable these enhancements. Bug fixes important to the correctness of coverage numbers are always enabled by default, with no need for -coverenhanced. Since the exact nature of -coverenhanced varies from release to release, the details of the enhancements it enables are present in the product release notes rather than in the Command Reference. For these details, search the release notes using the string "coverenhanced".

- -coverexcludedefault

Excludes VHDL code coverage data collection from the OTHERS branch in both Case statements and Selected Signal Assignment statements. Optional.

- -coveropt <opt_level>

Overrides the default level of optimization for the current run only. Optional. <opt_level> designates the optimization level, as follows:

1 — Turns off all optimizations that affect coverage reports.

2 — Allows optimizations that provide large performance improvements by invoking sequential processes only when the data changes. This setting may result in major reductions in coverage counts.

3 — Allows all optimizations in 2, and allows optimizations that may change expressions or remove some statements. Also allows constant propagation and VHDL subprogram inlining.

4 — Allows all optimizations in 2 and 3, and allows optimizations that may remove major regions of code by changing assignments to built-ins or removing unused signals. It also changes Verilog gates to continuous assignments. Allows VHDL subprogram inlining. Allows VHDL flip-flop recognition.

The default optimization level is 3. You can edit the [CoverOpt](#) variable in the *modelsim.ini* file to change the default.

- -coverreportcancelled

(optional) Enables code coverage reporting of branch conditions that have been optimized away due to a static or null condition. The line of code is labeled EA in the hits column of the

Source Window and EBCS in the hits column of a Coverage Report. You can also set this with the [CoverReportCancelled](#) *modelsim.ini* variable.

- **-coversub**
(optional) Re-enables code coverage data collection in VHDL subprograms previously disabled with `-nocoversub`. By default code coverage data is collected for VHDL subprograms. Edit the [CoverSub](#) variable in the *modelsim.ini* file to set a permanent default.
- **-nocoversub**
(optional) Disables code coverage data collection in VHDL subprograms. By default code coverage data is collected for VHDL subprograms. Edit the [CoverSub](#) variable in the *modelsim.ini* file to set a permanent default.
- **-debugVA**
Prints a confirmation if a VITAL cell was optimized, or an explanation of why it was not, during VITAL level-1 acceleration. Optional.
- **-deferSubpgmCheck**
Forces the compiler to report array indexing and length errors as warnings (instead of as errors) when encountered within subprograms. Subprograms with indexing and length errors that are invoked during simulation cause the simulator to report errors, which can potentially slow down simulation because of additional checking.
- **-dpiforceheader**
Forces the generation of a DPI header file even if it will be empty of function prototypes.
- **-error <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "error." Optional. Edit the [error](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-explicit**
Directs the compiler to resolve ambiguous function overloading by favoring the explicit function definition over the implicit function definition. Optional. Strictly speaking, this behavior does not match the VHDL standard. However, the majority of EDA tools choose explicit operators over implicit operators. Using this switch makes ModelSim compatible with common industry practice.
- **-extendedtogglemode 1|2|3**
Changes the level of support for extended toggles. Optional. The levels of support are:
 - 1 — 0L->1H & 1H->0L & any one 'Z' transition (to/from 'Z')
 - 2 — 0L->1H & 1H->0L & one transition to 'Z' & one transition from 'Z'
 - 3 — 0L->1H & 1H->0L & all 'Z' transitionsEdit the [ExtendedToggleMode](#) variable in the *modelsim.ini* file to set a permanent default.

- -f <filename>
Specifies a file with more command-line arguments. Optional. Allows complex argument strings to be reused without retyping. Allows gzipped input files. Nesting of -f options is allowed.
Refer to the section "[Argument Files](#)" for more information.
- -fatal <msg_number>[,<msg_number>,...]
Changes the severity level of the specified message(s) to "fatal." Optional. Edit the `fatal` variable in the `modelsim.ini` file to set a permanent default. Refer to "[Changing Message Severity Level](#)" for more information.
- -fsmimplicittrans
Enables recognition of implied same state transitions. Optional.
- -fsmmultitrans
Enables detection and reporting of multi-state transitions when used with the `+cover=f` argument for `vcom` or `vopt`. Optional. Another term for this is FSM sequence coverage.
- -fsmresettrans
Enables recognition of implicit asynchronous reset transitions. Optional. This includes asynchronous reset transitions in coverage results.
- -fsmsingle | -nofsmsingle
(optional) Enables the recognition of VHDL FSMs where the current state variable of type `std_logic`, `bit`, `boolean`, or single-bit `std_logic_vector/bit_vector` and Verilog single-bit FSMs.
- -fsmverbose [b | t | w]
Provides information about FSMs detected, including state reachability analysis. Optional. This switch only provides this data when you use the `-novopt` switch on the same command line.
 - b — displays only basic information.
 - t — displays a transition table in addition to the basic information.
 - w — displays any warning messages in addition to the basic information.

When you do not specify an argument, this switch reports all information similar to:

```
# ** Note: (vcom-1947)   FSM RECOGNITION INFO
#       Fsm detected in : ../fpu/rtl/vhdl/serial_mul.vhd
#       Current State Variable : s_state :
#       ../fpu/rtl/vhdl/serial_mul.vhd(76)
#       Clock : clk_i
#       Reset States are: { waiting , busy }
#       State Set is : { busy , waiting }
#       Transition table is
#
#       -----
#       busy      =>  waiting Line : (114 => 114)
#       busy      =>  busy    Line : (111 => 111)
#       waiting   =>  waiting Line : (120 => 120) (114 => 114)
#       waiting   =>  busy    Line : (111 => 111)
```

```
# -----
```

When you do not specify this switch, you will receive a message similar to:

```
# ** Note: (vcom-143) Detected '1' FSM/s in design unit
'serial_mul.rtl'.
```

- `-force_refresh <design_unit>`

Forces the refresh of all specified design units. Optional. By default, the work library is updated; use **-work <library_name>**, in conjunction with **-force_refresh**, to update a different library (for example, `vcom -work <your_lib_name> -force_refresh`).

When the compiler refreshes a design unit, it checks each dependency to ensure its source has not been changed and recompiled. Sometimes the dependency checking algorithm changes from release to release. This can lead to false errors during the integrity checks performed by the **-refresh** argument. An example of such a message follows:

```
** Error: (vsim-13) Recompile /u/test/dware/dware_61e_beta.dwpackages
because /home/users/questasim/linux/./synopsys.attributes has changed.
```

The **-force_refresh** argument forces the refresh of the design unit, overriding any dependency checking errors encountered by the **-refresh** argument.

A more conservative approach to working around **-refresh** dependency checks is to recompile the source code, if it is available.

- `-gen_xml <design_unit> <filename>`

Produces an XML-tagged file containing the interface definition of the specified entity. Optional. This option requires a two-step process where you must 1) compile `<filename>` into a library with **vcom** (without **-gen_xml**) then 2) execute **vcom** with the **-gen_xml** switch, for example:

```
vlib work
vcom counter.vhd
vcom -gen_xml counter counter.xml
```

- `-ignoredefaultbinding`

Instructs the compiler not to generate a default binding during compilation. Optional. You must explicitly bind all components in the design to use this switch.

- `-ignorevitalerrors`

Directs the compiler to ignore VITAL compliance errors. Optional. The compiler still reports that VITAL errors exist, but it will not stop the compilation. You should exercise caution in using this switch; as part of accelerating VITAL packages, we assume that compliance checking has passed.

- `-just abcep`

Directs the compiler to include only the following:

- a — architectures
- b — bodies

- c — configurations
- e — entities
- p — packages

Any combination in any order can be used, but you must specify at least one choice if you use this optional switch.

- -line <number>

Starts the compiler on the specified line in the VHDL source file. Optional. By default, the compiler starts at the beginning of the file.

- -lint

(optional) Performs additional static checks on case statement rules and enables warning messages for the following situations:

- The result of the built-in concatenation operator ("&") is the actual for a subprogram formal parameter of an unconstrained array type.
- If you specify the -BindAtCompile switch with vcom, the entity to which a component instantiation is bound has a port that is not on the component, and for which there is no error otherwise.
- A direct recursive subprogram call.
- In cases involving class SIGNAL formal parameters, as described in the IEEE Standard VHDL Language Reference Manual entitled "Signal parameters". This check only applies to designs compiled using -87. If you compile using -93, it would be flagged as a warning or error, even without the -lint argument. Can also be enabled using the [Show_Lint](#) variable in the *modelsim.ini* file.

- -maxfecrows

Sets the maximum number of rows allowed in an FEC truth table for a code coverage condition or expression. The default maximum is 192 rows, which allows for 96 terms in the expression. Increasing the number of rows includes more expressions for coverage, but also increases the compile time, sometimes dramatically. You can also configure this option using the [CoverMaxFECRows](#) variable in the *modelsim.ini* file.

- -maxudprows

Sets the maximum number of rows allowed in an UDP truth table for a code coverage condition or expression. The default maximum is 192 rows. Increasing the number of rows includes more expressions for coverage, but also increases the compile time, sometimes dramatically. You can also configure this option using the [CoverMaxUDPRows](#) variable in the *modelsim.ini* file.

- -mixedsvvh [b | l | r | i]

Facilitates using VHDL packages at the SystemVerilog-VHDL boundary of a mixed-language design. When you compile a VHDL package with -mixedsvvh, the package can be included in a SystemVerilog design as if it were defined in SystemVerilog itself. Optional.

- b — treats all scalars and vectors in the package as SystemVerilog bit type
- l — treats all scalars and vectors in the package as SystemVerilog logic type
- r — treats all scalars and vectors in the package as SystemVerilog reg type
- i — ignores the range specified with VHDL integer types

- **-modelsimini** <ini_filepath>

Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

- **-no1164**

Causes the source files to be compiled without taking advantage of the built-in version of the IEEE **std_logic_1164** package. Optional. This will typically result in longer simulation times for VHDL programs that use variables and signals of type **std_logic**.

- **-noaccel** <package_name>

Turns off acceleration of the specified package in the source code using that package.

- **-nocasestaticerror**

Suppresses case statement static warnings. Optional. VHDL standards require that case statement alternative choices be static at compile time. However, some expressions which are globally static are allowed. This switch prevents the compiler from warning on such expressions. If the **-pedanticerrors** switch is specified, this switch is ignored.

- **-nocheck**

Disables index and range checks. Optional. You can disable these individually using the **-noindexcheck** and **-norangecheck** arguments, respectively.

- **-nocoverfec**

Prevents focused expression coverage (FEC) from being enabled for coverage collection. By default, both UDP and FEC coverage statistics are enabled for collection. You can customize the default behavior with the [CoverFEC](#) variable in the *modelsim.ini* file. Optional.

- **-nocoverudp**

Prevents UDP coverage from being enabled for coverage collection. By default, both UDP and FEC coverage statistics are enabled for collection. You can customize the default behavior with the [CoverUDP](#) variable in the *modelsim.ini* file. Optional.

- **-nocoverrespecthandl**

Specifies that you want the VHDL 'H' and 'L' input values on conditions and expressions to be automatically converted to '1' and '0', respectively. By default in the current release, they are not automatically converted.

As an alternative to using this argument — if you are not using 'H' and 'L' values and don't want the additional UDP rows that are difficult to cover — you can either:

- Change your VHDL expressions of the form (a = '1') to (to_x01(a) = '1') or to std_match(a,'1'). These functions are recognized and serve to simplify the UDP tables
- Set the variable [CoverRespectHandL](#) in the *modelsim.ini* file to 0.

- -nocovershort

Disables short circuiting of expressions when coverage is enabled. Short circuiting is enabled by default. You can customize the default behavior with the [CoverShortCircuit](#) variable in the *modelsim.ini* file.

- -nodebug[=ports]

Hides, within the GUI and other parts of the tool, the internal data of all compiled design units. Optional.

-nodebug — The switch, specified in this form, does not hide ports, due to the fact that the port information may be required for instantiation in a parent scope.

The design units' source code, internal structure, registers, nets, etc. will not display in the GUI. In addition, none of the hidden objects may be accessed through the Dataflow or Schematic window or with commands. This also means that you cannot set breakpoints or single step within this code. It is advised that you not compile with this switch until you are done debugging.

Note that this is not a speed switch like the “nodebug” option on many other products. Use the [vopt](#) command to increase simulation speed.

-nodebug=ports — additionally hides the ports for the lower levels of your design; it should be used only to compile the lower levels of the design. If you hide the ports of the top level you will not be able to simulate the design.

Do not use the switch in this form when the parent is part of a [vopt](#) -bbox flow or for mixed language designs, especially for Verilog modules to be instantiated inside VHDL.

This functionality encrypts entire files. The `\protect` compiler directive allows you to encrypt regions within a file.

Design units or modules compiled with **-nodebug** can only instantiate design units or modules that are also compiled **-nodebug**.

- -nodbsym

Disables the generation of the symbols debugging database in the compiled library.

The symbols debugging database is the .dbs file in the compiled library that provides information to the GUI allowing you to view detailed information about design objects at the source level. Two major GUI features that use this database include source window annotation and textual dataflow.

You should only specify this switch if you know that anyone using the library will not require this information for design analysis purposes.

- **-noDeferSubpgmCheck**
Causes range and length violations detected within subprograms to be reported as errors (instead of as warnings). As an alternative to using this argument, you can set the `NoDeferSubpgmCheck` property in the `modelsim.ini` file to a value of 1.
- **-noindexcheck**
Disables checking on indexing expressions to determine whether indices are within declared array bounds. Optional.
- **-nofsmresettrans**
Disables recognition of implicit asynchronous reset transitions. Optional. This has the effect of excluding asynchronous reset transitions from any coverage results.
- **-nofsmxassign**
Disable recognition of FSMs containing x assignment. Optional.
- **-noFunctionInline**
Turns off VHDL subprogram inlining for design units using a local copy of a VHDL package. This may be needed in case the local package has the same name as an MTI supplied package.
- **-nologo**
Disables display of the startup banner. Optional.
- **-nonstddriverinit**
Forces ModelSim to match pre-5.7c behavior in initializing drivers in a particular case. Optional. Prior to 5.7c, VHDL ports of mode out or inout could have incorrectly initialized drivers if the port did not have an explicit initialization value and the actual signal connected to the port had explicit initial values. Depending on a number of factors, ModelSim could incorrectly use the actual signal's initial value when initializing lower level drivers. Note that the argument does not cause all lower-level drivers to use the actual signal's initial value. It does this only in the specific cases where older versions used the actual signal's initial value.
- **-noothersstaticerror**
Disables warnings that result from array aggregates with multiple choices having "others" clauses that are not locally static. Optional. If the **-pedanticerrors** switch is specified, this switch is ignored.
- **-nopsl**
Instructs the compiler to ignore embedded PSL assertions. By default **vcom** parses any PSL assertion statements it finds in the specified files. Refer to “[Compiling and Simulating Assertions](#)” for more information.

- **-norangecheck**
 Disables run time range checking. In some designs, this results in a 2X speed increase. Range checking is enabled by default or, once disabled, can be enabled using **-rangecheck**. Refer to “[Range and Index Checking](#)” for additional information.
- **-note <msg_number> [,<msg_number>, ...]**
 Changes the severity level of the specified message(s) to "note." Optional. Edit the [note](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-novital**
 Causes **vcom** to use VHDL code for VITAL procedures rather than the accelerated and optimized timing and primitive packages built into the simulator kernel. Optional. Allows breakpoints to be set in the VITAL behavior process and permits single stepping through the VITAL procedures to debug your model. Also all of the VITAL data can be viewed in the Locals or Objects windows.
- **-novitalcheck**
 Disables Vital level 1 checks and also Vital level 0 checks defined in section 4 of the Vital-95 Spec (IEEE Std 1076.4-1995). Optional.
- **-novopt**
 Forces **vcom** to produce code if the [VoptFlow](#) variable is set to 1 (optimizations turned on) in the *modelsim.ini*. (*VoptFlow* = 1 is the default behavior.) Optional. Use this argument together with the **vsim -novopt** command to run the simulator without any optimizations. One scenario in which you may want to use this switch is when coding an RTL block with a small testcase.
- **-nowarn <category_number>**
 Selectively disables a category of warning messages. Optional. Multiple **-nowarn** switches are allowed. Warnings may be disabled for all compiles via the Main window **Compile > Compile Options** menu command or the *modelsim.ini* file (Refer to [modelsim.ini Variables](#)).

The warning message categories are described in [Table 2-8](#):

Table 2-8. Warning Message Categories for vcom -nowarn

Category number	Description
1	unbound component
2	process without a wait statement
3	null range
4	no space in time literal
5	multiple drivers on unresolved signal

Table 2-8. Warning Message Categories for vcom -nowarn

Category number	Description
6	VITAL compliance checks (“VitalChecks” also works)
7	VITAL optimization messages
8	lint checks
9	signal value dependency at elaboration
10	VHDL-1993 constructs in VHDL-1987 code
11	PSL warnings
13	constructs that coverage can't handle
14	locally static error deferred until simulation run

- -O0 | -O1 | -O4 | -O5

Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

Please refer to the section "[Optimizing Designs with vopt](#)" in the User’s Manual for detailed information on using vopt to perform optimization.

- Enable PE-level optimization with **-O1**. Optional. Note that changing from the default **-O4** to **-O1** may cause event order differences in your simulation.
- Enable standard SE optimizations with **-O4**. Default. The main differences between **-O4** and **-O1** are that ModelSim attempts to improve memory management for vectors and accelerate VITAL Level 1 modules with **-O4**.
- Enable maximum optimization with **-O5**. Optional. **-O5** attempts to optimize loops and prevents variable assignments in situations where a variable is assigned but is not actually used. Using the **+acc** argument to **vcom** will cancel this latter optimization.
- -pedanticerrors
Forces display of an error message (rather than a warning) on a variety of conditions. Refer to "[Enforcing Strict 1076 Compliance](#)" for a complete list of these conditions. Optional. This argument overrides **-nocasestaticerror** and **-nootherstaticerror** (see above).
- -performdefaultbinding
Enables default binding when it has been disabled via the **RequireConfigForAllDefaultBinding** option in the *modelsim.ini* file. Optional.
- +protect [=<filename>]
Enables ``protect` and ``endprotect` compiler directives for encrypting selected regions of your design source code. Optional. Produces an encrypted output file with a *.vhdp* extension in the default work directory. To create an encrypted output file to the current directory, add

=<filename> to this argument. If you specify a filename, all source files on the command line are concatenated together into a single output file.

Any include files will also be inserted into the output file when you add =<filename>. If you do not use =<filename>, all include files will be encrypted into the work directory as individual files, not merged together into one file.

- **-pslxt**
Enables PSL LTL and OBE operators. Optional. These operators are disabled by default.
- **-pslfile <filename>**
Identifies an external PSL assertion file to compile along with the VHDL source files. Refer to “[Compiling and Simulating Assertions](#)” for more information.
- **-quiet**
Disables 'Loading' messages. Optional.
- **-rangecheck**
Enables run time range checking. Default. Range checking can be disabled using the **-norangecheck** argument. Refer to “[Range and Index Checking](#)” for additional information.
- **-refresh**
Regenerates a library image. Optional. By default, the work library is updated. To update a different library, use **-work <library_name>** with **-refresh** (for example, `vcom -work <your_lib_name> -refresh`). If a dependency checking error occurs which prevents the refresh, use the **vcom -force_refresh** argument. See the **vcom** Examples for more information. You may use a specific design name with **-refresh** to regenerate a library image for that design, but you may not use a file name.
- **-s**
Instructs the compiler not to load the **standard** package. Optional. This argument should only be used if you are compiling the **standard** package itself.
- **-skip abcep**
Directs the compiler to skip all:
 - a — architectures
 - b — bodies
 - c — configurations
 - e — entities
 - p — packagesAny combination in any order can be used, but one choice is required if you use this optional switch.

- **-source**
Displays the associated line of source code before each error message that is generated during compilation. Optional. By default, only the error message is displayed.
- **-suppress <msg_number>[,<msg_number>,...]**
Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress Fatal or Internal messages. Edit the [suppress](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-synthprefix <prefix>**
(optional) Enables recognition of synthesis pragmas with a user specified prefix. If this argument is not specified, pragmas are treated as comments and the previously excluded statements included in the synthesized design. All regular synthesis pragmas are honored.
<prefix> — Specifies a user defined string.
- **-time**
Reports the "wall clock time" **vcom** takes to compile the design. Optional. Note that if many processes are running on the same system, wall clock time may differ greatly from the actual "cpu time" spent on **vcom**.
- **-togglecountlimit <int>**
Limits the toggle coverage count, <int>, for a toggle node. Optional. After the limit is reached, further activity on the node is ignored for toggle coverage. All possible transition edges must reach this count for the limit to take effect. For example, if you are collecting toggle data on 0->1 and 1->0 transitions, both transition counts must reach the limit. If you are collecting "full" data on 6 edge transitions, all 6 must reach the limit. Overrides the global value set by the [ToggleCountLimit](#) *modelsim.ini* variable.
- **-togglewidthlimit <int>**
Sets the maximum width of signals, <int>, that are automatically added to toggle coverage with the **-cover t** argument. Optional. Can be set on design unit basis. Overrides the global value of the [ToggleWidthLimit](#) *modelsim.ini* variable.
- **-version**
Returns the version of the compiler as used by the licensing tools. Optional.
- **-vmake**
Generates a complete record of all command line data and files accessed during the compile of a design. This data is then used by the [vmake](#) command to generate a comprehensive makefile for recompiling the design library. By default, vcom stores compile data needed for the **-refresh** switch and ignores compile data not needed for **-refresh**. The **-vmake** switch forces inclusion of all file dependencies and command line data accessed during a compile, whether they contribute data to the initial compile or not. Executing this switch can increase compile time in addition to increasing the accuracy of the compile. See the [vmake](#) command for more information.

- **-vopt**
Notifies **vcom** that the **vopt** command will be run. As a result, **vcom** does not produce code. This saves an unnecessary code generation step. Only needed if **VoptFlow** is set to 0 in the *modelsim.ini*. If **VoptFlow** is set to 1, the **vcom** code generation step is skipped automatically. Optional.
- **-warning <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "warning." Optional. Edit the **warning** variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-work <library_name>**
Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional; by default, the compiled design units are added to the **work** library. The specified pathname overrides the pathname specified for work in the project file.
- **<filename>**
Specifies the name of a file containing the VHDL source to be compiled. One filename is required; multiple filenames can be entered separated by spaces or wildcards may be used (e.g., *.vhd).

If you don't specify a filename, and you are using the GUI, a dialog box pops up allowing you to select the options and enter a filename.

Examples

- Compile the VHDL source code contained in the file *example.vhd*.

```
vcom example.vhd
```
- ModelSim supports designs that use elements conforming to the 1987, 1993, and 2002 standards. Compile the design units separately using the appropriate switches.

```
vcom -87 o_units1.vhd o_units2.vhd  
vcom -93 n_unit91.vhd n_unit92.vhd
```
- Hide the internal data of *example.vhd*. Models compiled with **-nodebug** cannot use any of the ModelSim debugging features; any subsequent user will not be able to see into the model.

```
vcom -nodebug example.vhd
```
- The first line compiles and hides the internal data, plus the ports, of the lower-level design units, *level3.vhd* and *level2.vhd*. The second line compiles the top-level unit, *top.vhd*, without hiding the ports. It is important to compile the top level without **=ports** because top-level ports must be visible for simulation.

```
vcom -nodebug=ports level3.vhd level2.vhd  
vcom -nodebug top.vhd
```


- When compiling source that uses the **numeric_std** package, this command turns off acceleration of the **numeric_std** package, located in the **ieee** library.

```
vcom -noaccel numeric_std example.vhd
```

- Although it is not obvious, the = operator is overloaded in the **std_logic_1164** package. All enumeration data types in VHDL get an “implicit” definition for the = operator. So while there is no explicit = operator, there is an implicit one. This implicit declaration can be hidden by an explicit declaration of = in the same package (LRM Section 10.3). However, if another version of the = operator is declared in a different package than that containing the enumeration declaration, and both operators become visible through **use** clauses, neither can be used without explicit naming.

```
vcom -explicit example.vhd
```

To eliminate that inconvenience, the VCOM command has the **-explicit** option that allows the explicit = operator to hide the implicit one. Allowing the explicit declaration to hide the implicit declaration is what most VHDL users expect.

```
ARITHMETIC."="(left, right)
```

- The **-work** option specifies **mylib** as the library to regenerate. **-refresh** rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of ModelSim (4.6 and later only).

```
vcom -work mylib -refresh
```

- The **-fsmmultitrans** option enables detection and reporting of multi-state transitions when used with the **+cover f** argument.

```
vcom +cover=f -fsmmultitrans
```

vcover attribute

This command is used to display attributes in the currently loaded database, during batch mode simulation, on the following types of attributes:

- Test Attributes — sets the value of attributes for testcase information. Refer to the section "[Predefined Attribute Data](#)" for complete list of these attributes.

Syntax

For test attributes

```
vcover attribute <file> [-test <testname>] [-tcl] [-concise]
[-modelsimini <ini_filepath>]
[-name <attribute> ...]
[-suppress <msg_number>[,<msg_number>,...]]
[-warning <msg_number>[,<msg_number>,...]]
[-error <msg_number>[,<msg_number>,...]]
[-note <msg_number>[,<msg_number>,...]]
```

Arguments

- <file>
(required) The database you want to analyze.
- -concise
(optional) Print attribute values only, do not print other information.
- -modelsimini <ini_filepath>
(optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- -name <attribute> ...
(optional) Reports data for the specified attribute. You can specify this option any number of times.
- -tcl
(optional) Prints attribute information in a tcl format.
- -test <testname>
(optional) Reports attribute data for the specified testname. This is most useful when reporting on merged UCDB files that contain many tests.
- -error <msg_number>[,<msg_number>,...]
Changes the severity level of the specified message(s) to "error." Optional. Does not function with internal messages (those without numbers). Edit the [error](#) variable in the *modelsim.ini* file to set a permanent default. Refer to "[Changing Message Severity Level](#)" for more information.

- `-note <msg_number>[,<msg_number>,...]`
Changes the severity level of the specified message(s) to "note." Optional. Does not function with internal messages (those without numbers). Edit the [note](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- `-suppress <msg_number>[,<msg_number>,...]`
Prevents the specified message(s) from displaying. The `<msg_number>` is the number preceding the message you wish to suppress. Optional. You cannot suppress Fatal or Internal messages (those without numbers). Edit the [suppress](#) variable in the *modelsim.ini* file to set a permanent default.
- `-warning <msg_number>[,<msg_number>,...]`
Changes the severity level of the specified message(s) to "warning." Optional. Does not function with internal messages (those without numbers). Edit the [warning](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

Examples

- Report all attributes of the file test.ucdb
vcover attribute test.ucdb
- Report only the USERNAME and HOSTNAME attributes for the file test.ucdb
vcover attribute test.ucdb -name USERNAME -name HOSTNAME

Related Topics

- [Verification Browser Window](#)
- [Verification Tracker Window](#)
- [coverage attribute](#)
- [coverage exclude](#)
- [coverage goal](#)
- [coverage ranktest](#)
- [coverage report](#)
- [coverage save](#)
- [coverage tag](#)
- [coverage analyze](#)
- [coverage testnames](#)
- [coverage weight](#)
- [vcover merge](#)
- [vcover ranktest](#)
- [vcover stats](#)

vcover dump

This command produces and prints to *stdout* a textual description of the contents of the UCDB file. Output may be inconsistent from release to release.

Syntax

```
vcover dump <ucdb_file> [-file <dump_text_file>] [-canonical]
  [-suppress <msg_number>[,<msg_number>,...]]
  [-error <msg_number>[,<msg_number>,...]]
  [-note <msg_number>[,<msg_number>,...]]
  [-warning <msg_number>[,<msg_number>,...]]
```

Arguments

- <ucdb_file>
(required) The name of the UCDB file to be printed out.
- -file <dump_text_file>
(optional) Prints the text output to a specified file, instead of to *stdout*.
- -canonical
(optional) Sorts nodes and tags at each level of the UCDB. Presents output in an organized, predictable manner, enabling easier comparison of multiple *.ucdb* files. It is limited to 1024 levels, and is less memory efficient than an ordinary dump.
- -error <msg_number>[,<msg_number>,...]
Changes the severity level of the specified message(s) to "error." Optional. Does not function with internal messages (those without numbers). Edit the [error](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- -note <msg_number>[,<msg_number>,...]
Changes the severity level of the specified message(s) to "note." Optional. Does not function with internal messages (those without numbers). Edit the [note](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- -suppress <msg_number>[,<msg_number>,...]
Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress Fatal or Internal messages (those without numbers). Edit the [suppress](#) variable in the *modelsim.ini* file to set a permanent default.
- -warning <msg_number>[,<msg_number>,...]
Changes the severity level of the specified message(s) to "warning." Optional. Does not function with internal messages (those without numbers). Edit the [warning](#) variable in the

modelsim.ini file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

Examples

- Print a textual output of the file *test.ucdb*
vcover dump test.ucdb
- Print a canonical version of the output of file *test.ucdb*
vcover attribute -file test.text -canonical test.ucdb

Related Topics

- [vcover merge](#)
- [vcover ranktest](#)
- [vcover stats](#)

vcover merge

The **vcover merge** command merges multiple code or functional coverage data files that were created with the [coverage save](#) command. The command adjusts its functionality to the type of coverage files (functional coverage or code coverage) given on its command line. All files being merged must have been created from the same design.

By default, vcover merge creates a test-associated merge, which associates coverage items with the test(s) that covered them. You can then analyze the results of this merge using the [coverage analyze](#) command. To obtain a more basic level of information use the **-totals** argument without **-test**.

There are cases in which it may be advisable not to merge, and instead preserve the individual UCDBs for analysis and ranking. In the following cases, the tool issues a warning message indicating that the resulting merged file cannot completely represent the merged information:

- if "at_least" is greater than 1
- weights are different for the same object in different files
- different objects in different files

For these, you can run with the **-verbose** argument set to obtain further details about potential issues with the merge. For more information, see "[Limitations of Merge for Coverage Analysis](#)".

If a code coverage instance in the unified coverage database (UCDB) has been changed, a warning will be generated. Warnings can be disabled with the **-quiet** option.

The command can be invoked within the ModelSim GUI or at the system prompt.

Syntax

```
vcover merge <merge_options> [-out <outfile>] <file1> [<file2> ...<filen>]
```

<merge_options> =

```
[-and] [-append] [-backup] [-inputs <file>] [-install <path>]  
[[-instance <path> [-recursive]] | [-du <du_name> [-recursive]]]  
[-ignoredusig] [-lockingfull | -lockingtimeout | -lockingnone] [-log <filename>]  
[-modelsimini <ini_filepath>] [-notagging] [-strip <n>] [-showambiguity] [-quiet]  
[-verbose] [-version] [-combine | -totals | -testassociated]  
[-timeout <seconds>] <file1> [<file2> ...<filen>]  
[-error <msg_number>[,<msg_number>,...]]  
[-note <msg_number>[,<msg_number>,...]]  
[-suppress <msg_number>[,<msg_number>,...]]  
[-warning <msg_number>[,<msg_number>,...]]
```

Arguments

- -and
Excludes statements in the output file *only* if they are excluded in all input files. Optional. By default a statement is excluded in the output merge file if the statement is excluded in any of the input files. Not a valid option when merging functional coverage databases.
- -append
Specifies that progress messages are to be appended to the current log file. Optional. By default a new log file is created each time you invoke the command. Not a valid option when merging functional coverage databases.
- -backup
Creates a backup UCDB output file named "< ucdb filename >._backup" during the lock-protected execution of vcover merge.
- -combine
Merges two or more different runs of a single test, or re-joining stripped versions of a UCDB file. When using this argument, for nodes with conflicting toggle information, both a minimum and maximum count is saved in UCDB. Only minimum counts are saved for conflicting non-toggle data. Mutually exclusive with -totals and -testassociated.
- -du <du_name> [-recursive]
Instructs the tool to merge all instances of the specified design unit in all the input files. It then creates an output file consisting of one instance of the design unit, containing all the merged data. <du_name> is [<library name>.<primary>[(<secondary>)]], where the library name is optional, and secondary name is required only for VHDL.
Instance names in the output file are generated in the following format, replacing any '/' with '_' from the library names:
`<library>_<primary>[_<secondary>]`
The -recursive argument instructs the tool to merge the complete design subtree, from the designated design unit down. Optional. If not specified, just the level specified is merged.
- -error <msg_number>[,<msg_number>,...]
Changes the severity level of the specified message(s) to "error." Optional. Does not function with internal messages (those without numbers). Edit the `error` variable in the `modelsim.ini` file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- -ignoreusig
Instructs the tool to ignore design unit signature checking and continue merging. This argument should not be used lightly, without first validating that the differences in code between the merges of two versions of the same file are expected and approved. See “[Merging and Source Code Mismatches](#)” for further details on the use of this argument.

- **-inputs <file>**
Specifies a text file containing input filenames that you want to merge. Optional.
- **-instance <path> [-recursive]**
Instructs the tool to merge all occurrences of the specified instances in all the input files. It then creates an output file consisting of a single instance, containing all the merged data.
You can change the resulting path using the **-install <path>** option.
The **-recursive** argument instructs the tool to merge the complete design subtree, from the designated instance down. Optional. If not specified, just the level specified is merged.
- **-install <path>**
Adds <path> as additional hierarchy on the front end of instance and object names in the data files. Optional. This argument allows you to merge coverage results from simulations that have different hierarchies. See "[Merge Usage Scenarios](#)" for more information.
- **-lockingfull | -lockingtimeout | -lockingnone**
Controls the locking of the merge file. Optional.
 - lockingfull — (default) Supports breaking of locks when the creating PID is dead or when a time limit has been passed. Requires “rsh” to the machine that created the merge.
 - lockingtimeout — Supports breaking of locks when a time limit has been passed. Does not require “rsh” to creating machine. See note below for asynchronous clocks.
 - lockingnone — No locking and faster performance.

Note



The **-lockingtimeout** version of timeout checking is only accurate when machine clocks are synchronized.

- **-log <filename>**
Specifies the file for outputting progress messages. Optional. By default these messages are output to *vcover.log*.
- **-modelsimini <ini_filepath>**
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-notagging**
Prevents the automatic implicit test plan tagging from being performed.
- **-note <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "note." Optional. Does not function with internal messages (those without numbers). Edit the [note](#) variable in the

modelsim.ini file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

- **-out <outfile>**
Specifies the name of the file that will contain the merged output. Optional. When **-out** is not specified, the filename used for the output is the base name of the first UCDB file listed in the command, whereas when **-out** is specified, the output file can be specified anywhere in the command.
- **-quiet**
Disable warnings when merging databases and a changed instance is encountered.
- **-showambiguity**
When used, **vcover merge** displays both minimum and maximum counts for any conflicting toggle data in a UCDB that results from a combined merge.
- **-strip <n>**
Removes <n> levels of hierarchy from instance and object names in the data files. Optional. This argument allows you to merge coverage results from simulations that have different hierarchies. See "[Merge Usage Scenarios](#)" for more information.
- **-suppress <msg_number>[,<msg_number>,...]**
Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress Fatal or Internal messages (those without numbers). Edit the [suppress](#) variable in the *modelsim.ini* file to set a permanent default.
- **-testassociated**
Merges the selected databases, including all the basic information (created with **-totals**) as well as the associated tests and bins. This is the default merge. This argument is mutually exclusive with **-totals** and **-combine**.

When tests and bins are associated, each coverage count is marked with the test that caused it to be covered. For functional coverage, this means the bin count should be greater than or equal to the `at_least` parameter. For code coverage and assertion data, any non-zero count for a test causes the bin to be marked with the test. While it cannot be known which test incremented a bin by exactly how much, it can be known which test caused a bin to be covered.
- **-totals**
Merges the databases with a basic level of information, including: coverage scopes, design scopes, and test plan scopes. The counts are incremented together. In the case of vector bin counts, counts are ORed. The final output database is a union of objects from the input files. Information about which test contributed what coverage into the merge is lost. Information about tests themselves are not lost — test data records are added together from all merge inputs. While the list of tests can be known, it cannot be known what tests might have incremented particular bins. Mutually exclusive with **-combine** and **-testassociated**.

- **-timeout <seconds>**

Sets the timeout period after which the lock can be removed. During the timeout period the lock holder is protected. After the timeout expires, it is open hunting season. Supports cumulative merges and multiple merge commands, issued one after another. Such merge commands can be issued simultaneously from various platforms in a networking environment. In order to avoid corrupting cumulative coverage results, merges of UCDB files are serialized.
- **-verbose**

Enables summary code coverage statistics to be computed and directed to the log file each time a file is merged into the base. The statistics are instance-based. Optional. Not a valid option when merging functional coverage databases.
- **-version**

Returns the version number of each input UCDB file, and the version number of the output UCDB file, which is always created with the most recent version of the UCDB creation software. Optional.
- **-warning <msg_number>[,<msg_number>,...]**

Changes the severity level of the specified message(s) to "warning." Optional. Does not function with internal messages (those without numbers). Edit the [warning](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **<file1> [<file2> ...<fileN>]**

Specifies the file(s) you want to merge, with <file1> required to contain the superset of the objects to be merged. Subsequent files listed must be subsets of the first file listed. Required. Multiple pathnames and wildcards are allowed. See Examples.

Examples

- Merge coverage statistics for *myfile1* and *myfile2* and writes them to *myresult*.

```
vcover merge myfile1 myfile2 -out myresult
```

- Use wildcards to merge all files with a *.cov* extension in a particular directory.

```
vcover merge myresult2 /dut/*.cov
```

- Create a trend database to be used with vcover report for trending analysis, saving the file to *mytrend.ucdb*.

```
vcover merge -trend -output mytrend.ucdb Nov_mg.ucdb Dec_mg.ucdb
```

The *Nov_mg.ucdb* and *Dec_mg.ucdb* are two merged files from simulations of the same DUT.

- Change the default time-out for the lock file to 600 seconds.

```
vcover merge -timeout 600 out1.ucdb out2.ucdb in.ucdb
```

- Strip the top two levels of hierarchy from an instance or objects in *myfile.ucdb* and place into another file *myfile_stripped.ucdb*.

```
vcover merge -strip 2 myfile_stripped.ucdb myfile.ucdb
```

See also

[“Code Coverage”](#), [“Verification with Functional Coverage”](#), [“Verification with Assertions and Cover Directives”](#), ["Merge Usage Scenarios"](#), [coverage analyze](#), [coverage attribute](#), [coverage ranktest](#), [coverage save](#), [coverage tag](#), [coverage testnames](#), [coverage unlinked](#), [vcover attribute](#), [vcover merge](#), [vcover ranktest](#), [vcover stats](#)

vcover ranktest

The **vcover ranktest** command ranks coverage data contained in the specified tests, according to their contribution to cumulative coverage. The tests to be ranked are taken from the UCDB files specified as inputs. These can be merged or unmerged UCDBs. If merged, they must have been created without the vcover merge **-totals** argument.

The output of this command is written to stdout (screen) and two lists of tests (contributing *ranktest.contrib* and non-contributing tests in *ranktest.noncontrib*). The order of the tests written to the screen or a specified file (with **-log**) are listed in the order shown in [Table 2-9](#).

Table 2-9. Order and Type of Ranked Tests

Contributing, compulsory	Mandatory tests, tests which need to be run regardless of achieved coverage.	Sorted by total coverage %
Contributing, noncompulsory	Tests providing coverage not provided by any previous test.	Sorted by total coverage %
Non-contributing	Redundant tests, providing no incremental coverage.	Not sorted

To rank on a specific coverage item, design unit, or test plan section within the hierarchy of your design, use **-path**, **-du** or **-plansection**.

Syntax

```
vcover ranktest <ranktest_options> {<UCDB_inputfile1> [... <UCDB_inputfileN>]}
```

<ranktest_options> =

```
[-inputs <file_list>]
[-log <filename>]
[-assertion] [-assertfailure] [-code {b | c | e | f | s | t}...] [-codeAll] [-cvg] [-directive]
[-nocompulsoryordering | -compulsorysorting | -compulsoryranking]
[-goal [<coverage_type> <percentage>]]
[-iterative | -testassociated]
[-maxcpu <real_num_in_seconds>] [-maxtests <int>] [-modelsimini <ini_filepath>]
[-metric {aggregate | total}]
[-fewest | -cputime | -simtime]
[-path <path> | -du <du_name> | -plansection <path>]
[-keepmergefile <filepath>]
[-precision <int_num>]
[-rankfile <filename>]
[-quiet | -concise | -verbose]
[-weight <coverage_type> <integer>]
[-error <msg_number>[,<msg_number>,...]] [-note <msg_number>[,<msg_number>,...]]
[-suppress <msg_number>[,<msg_number>,...]]
[-warning <msg_number>[,<msg_number>,...]]
```

Arguments

- -assertion

Specifies ranktest for assertion data. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.

Traditionally, assertions are a measure of correctness and not a measure of coverage. When viewed as a coverage item, their definitions are as follows:

- Percent coverage assertion passes (-assertion): Of all assertions what percent were evaluated to completion and had at least one non-vacuous pass.
- Percent coverage assertion failures (-assertfailures): Of all assertions what percent were evaluated to completion and had at least one failure. This is **THE ONLY NEGATIVE COVERAGE VALUE** in the sense that 0% coverage is desirable and 100% coverage is very undesirable. As such, it is highly recommended that -assertfailures only be used in a mutually exclusive fashion with the other coverage values.

It is possible for any given assertion to have both passes and failures. It is also possible for any given assertion to not ever be evaluated to completion. As such, often the sum of assertion passes and assertion failures will not be 100%.

- -assertfailure

Specifies ranktest for failed assertion data only. See [-assertion](#) for more information. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.

- -code {b | c | e | f | s | t}...

Specifies ranktest for corresponding code coverage type only: branch, condition, expression, statement, toggle, FSM. More than one coverage type can be specified with each -code argument (example: “-code bcesf”). Optional. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.

- -codeAll

Specifies ranktest for all coverage types. Equivalent to -code bcestf. Optional. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.

- -concise

Specifies the output is created with minimum additional I/O. Optional. Default creates ranktest with full I/O (-verbose). Mutually exclusive with -quiet and -verbose.

- [-nocompulsoryordering](#) | -compulsorysorting | -compulsoryranking

Specifies how the command ranks compulsory tests, when present.

-nocompulsoryordering — (default) Ranks compulsory tests in a random order with no ranking with respect to each other. Does not display compulsory tests in the detailed ranktest listing. In the GUI, the accumulated coverage data and incremental coverage

data for compulsory tests is displayed as a minus sign (-). This option has the best performance of the three options.

-compulsorysorting — Ranks compulsory tests in the order of descending 'total coverage' with respect to each other. Displays compulsory tests in the detailed ranktest listing according to this order. In the GUI, the accumulated coverage data and incremental coverage data for compulsory tests is calculated and displayed. This option has performance second to the -nocompulsoryordering option.

-compulsoryranking — Ranks compulsory tests fully with respect to each other. Displays compulsory tests in the detailed ranktest listing according to this order. In the GUI, the accumulated coverage data and incremental coverage data for compulsory tests will be calculated and displayed. This option can have the worst performance of the three options.

- -cvg
Specifies ranktest for covergroup data. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.
- -cputime
Specifies that the tests be ranked by minimum CPU time. Optional. Mutually exclusive to the **-fewest** and **-simtime** arguments. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.
- -directive
Specifies ranktest for directive data. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.
- -du <du_name>
Restricts the ranking to the specified design unit. Optional. This argument applies to a particular module type, by name, in all UCDB files. This argument is not compatible with **-iterative**: if both are explicitly set, ranking is halted and an error is issued. This option is mutually exclusive with **-path** and **-plansection**.
- -error <msg_number>[,<msg_number>,...]
Changes the severity level of the specified message(s) to "error." Optional. Does not function with internal messages (those without numbers). Edit the **error** variable in the *modelsim.ini* file to set a permanent default. Refer to [“Changing Message Severity Level”](#) for more information.
- -fewest
Specifies that the files be ranked by fewest number of tests. Optional. Mutually exclusive to the **-cputime** and **-simtime** arguments. Default.
- -inputs <file_list>
Specifies a file containing ranktest arguments. Optional. The file can contain a list of UCDB files to be ranked.

- **-iterative**
Ranks the coverage items in the specified database(s) with a basic level of information, including: coverage scopes, design scopes, and test plan scopes. Optional. Mutually exclusive with **-testassociated**.
- **-goal [<coverage_type>] <percentage>**
Specifies that the tests be ranked to achieve this coverage goal. Optional. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.

The <coverage_type> specifies that the goal is applied only to the specified type of coverage; otherwise, the goal is applied to all types of coverage. Valid values for <coverage_type> are: **-assertion**, **-assertfailure**, **-cvg**, **-directive**, **-code {b | c | e | f | s | t}...** or **-codeAll**.

<percentage> is an integer value, whose default is 100.

The **-goal** argument can be specified multiple times, as shown in this example:

```
vcover ranktest -goal -assertion 90 -goal -code bcest 95
```
- **-keepmergefile <filepath>**
Specifies the merge file corresponding to the ranking be preserved. By default, the merge file is deleted.
- **-log <filename>**
Specifies the file for outputting ranked results. Output includes full path to tests. Optional.
- **-maxcpu <real_num_in_seconds>**
Monitors the accumulated CPU time of the ranked tests. Specifies the maximum CPU time to be allowed. If the specified number of seconds is exceeded, the ranking process is stopped. The default value is -1.0 (no limit). Optional.
- **-maxtests <int>**
Specifies threshold for the maximum number of tests to be ranked. When this threshold is exceeded, the ranking operation is terminated. Optional.
- **-metric {aggregate | total}**
Indicates the kind of metric used for ranking. Optional. Arguments:
total — (Default, unless any of the following arguments are used: **-goal**, **-weight**, **-assertion**, **-assertfailures**, **-cvg**, **-directive**, **-codeAll**, or **-code**)
Ranking metric used produces values consistent with the totals for (verification) test plan sections obtained with [coverage analyze](#).

aggregate —
Ranking metric used produces aggregate values based on each individual coverage type: values are *not* likely to be consistent with totals produced with [coverage analyze](#). See “Important” note below.

Each coverage type can be selected or not. Each coverage type can be given an individual weight and goal. These individual numbers are then combined and normalized to yield an aggregate metric that is unrelated to the number given by the total coverage.

i **Important:** When the **-metric aggregate** argument is used, the resulting metric number will not “match” any other total coverage number produced by other verification tools (i.e. coverage analyze). This is important because when you use any of the arguments (**-totals**, **-goal**, **-weight**, **-assertion**, **-assertfailures**, **-cvg**, **-directive**, **-codeAll**, or **-code**) with ranktest command, the aggregate metric is the default.

- **-modelsimini <ini_filepath>**
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-note <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "note." Optional. Does not function with internal messages (those without numbers). Edit the **note** variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-path <path>**
Restricts ranking to design paths (non-testplan) matching the specified <path>. Optional. This argument applies to a sub-hierarchy in all UCDB files. This option is mutually exclusive with **-du** and **-plansection**.
- **-plansection <path>**
Restricts ranking to the specified testplan node. Optional. This argument applies to a particular module type, by name, in all UCDB files. This argument is not compatible with **-iterative**: if both are explicitly set, ranking is halted and an error is issued. This option is mutually exclusive with **-du** and **-path**.
- **-precision <int_num>**
Specifies the decimal point precision for output only: The contents of the rank file are NOT affected by this argument. <int_num> is an integer value. The default value is 2.
- **-quiet**
Creates the ranktest output without any additional I/O. Optional. Default creates ranktest with full I/O (**-verbose**). Mutually exclusive with **-concise** and **-verbose**.
- **-rankfile <filename>**
Specifies the name of the ranktest file being created. Optional. Default if not specified is *ranktest.rank*. Can be specified with the **vcover stats** command to redisplay the results of this ranking. This file can also be used to repopulate the Browser with ranktest information.

- **-simtime**
Specifies that the tests be ranked by minimum simulation time. Optional. Mutually exclusive to the **-cputime** and **-fewest** arguments.
- **-suppress** <msg_number>[,<msg_number>,...]
Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress Fatal or Internal messages (those without numbers). Edit the [suppress](#) variable in the *modelsim.ini* file to set a permanent default.
- **-testassociated**
Ranks the coverage items in the selected database(s) including all the basic information (as created with **-iterative**) as well as the associated tests and bins. Optional. This is the default ranktest. This argument is mutually exclusive with **-iterative**.
- **-verbose**
Specifies the output is created with full I/O. Optional. Mutually exclusive with **-quiet** and **-concise**. This is the default.
- **-warning** <msg_number>[,<msg_number>,...]
Changes the severity level of the specified message(s) to "warning." Optional. Does not function with internal messages (those without numbers). Edit the [warning](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-weight** <coverage_type> <integer>
Used when selecting next ranking candidate. Optional. This argument is not compatible with **-metric total**: if both are explicitly set, ranking is halted and an error is issued.
Valid values for <coverage_type> are: **-assertion**, **-assertfailure**, **-code {b | c | e | f | s | t}...**, **-codeAll**, **-cvg**, or **-directive**.
The **-weight** argument can be specified multiple times, as shown in this example:

```
vcover ranktest -weight -assertion 10 -weight -code bcest 5
```
- <UCDB_inputfile1> [... <UCDB_inputfileN>]
Specifies the name of two or more non-merged UCDB file(s) to rank. Required, unless the **-inputs** file is used, specifying one or more UCDB files to be ranked.

See also

[vcover merge](#), “[Code Coverage](#)”, [coverage goal](#), [coverage weight](#), [coverage ranktest](#)

vcover report

The **vcover report** command prints textual output of coverage statistics or exclusions — from a previously saved code or functional coverage run — to a specified file. This allows you to produce reports "offline" (i.e., without having to load a simulation.)

You can choose from a number of report output options using the arguments listed below.

By default, the command prints out results from the current scope. To specify a certain path for the report, you can use the **-instance** argument, or specify the specific cover directive or covergroup, such as:

- `vcover report -instance <path>`, or
- `vcover report <path_to_cover_directive/covergroup>`

The command orders the output on a by-file basis unless you specify the **-byinstance** or **-bydu** argument.

The report displays code coverage data from generate blocks.

Syntax

```
vcover report [<coverage_arguments>] -file <filename>
```

```
vcover report [-version]
```

Global Arguments - Usable with any other arguments

```
vcover report [-above <percent> | -below <percent>] [-append] [-binrhs]  
[-details [-dumptables] [-fecanalysis] [-metricanalysis]]  
[-memory] [-modelsimini <ini_filepath>] [-zeros] [-nomissing]  
[-nozeroweights] [-precision <int>] [-recursive [-depth <n>]]  
[-showambiguity]  
[-testextract <test_name_or_pattern>] [-file <filename>] [-xml]  
[-suppress <msg_number>[,<msg_number>,...]]  
[-error <msg_number>[,<msg_number>,...]]  
[-note <msg_number>[,<msg_number>,...]]  
[-warning <msg_number>[,<msg_number>,...]]
```

Create HTML output from a UCDB

```
vcover report [-html [-code [bcesf[t|x]]] [-assert] [-cvg] [-directive] [-verbose] [-nosource] [-  
noframes] [-nodetails] [-summary] [-htmldir <outdir>] [-threshL <val>] [-threshH <val>]  
<input_ucdb>]
```

Filtering Arguments - Used to filter one or more coverage types in the report

```
vcover report [-assert] [-code {b | c | e | f | s | t}...] [-concurrent | -immediate] [-covered] [-cvg]  
[-directive] [-lang sva | psl | vhdl] [-severity info | note | warning | error | failure | fatal]  
[-testattr] [-unattemptedimmed]
```

For each directive instance, the report includes by default the full instance path, the coverage count or percentage, the design unit, the source file name, and the source line number.

Code Coverage Arguments - Has no effect on assertion or functional coverage

```
vcover report [-bydu] [-byfile] [-byinstance] [-coverenhanced] [-totals] [-noannotate]
  [-library <libname>] [-du <du_name>] [-package <pkgname>]
  [-source <filename>] [-instance <path> [-recursive [-depth <n>]]]
```

Exclusion-specific Coverage Arguments

```
vcover report [-excluded [-pragma | -user]] [-noexcludedhits]
```

Toggle-specific Coverage Arguments

```
vcover report [-toggles] [-duplicates] [-verbose] [-all] [-top]
  [-select {inputs | outputs | inout | ports | internals}]
```

Toggle coverage statistics are relevant only when reporting on instances or design units and are not produced on a per file basis. Toggle data is summed for all instances, and is reported by port or local name in the design unit, rather than by the connected signal. If you want toggle coverage statistics, you must specify either the **-byinstance**, **-bydu**, **-instance <path>**, or **-du <du_name>** arguments. If you do not use those arguments, or you use the **-source <filename>** argument, toggle coverage statistics are excluded even if you specify **-code t**. To get an itemized list of the signals, the **-details** argument is also required.

To report extended toggle coverage, ensure that you have compiled (vlog/vcom) with the **-code x** argument, then use **vcover report** with **-code t**.

Covergroup-specific Arguments

```
vcover report [-noignorebin] [-option] [-samples <spec>] [-usecnpm]
```

Arguments

- **-above <percent>**
Specifies that only objects with coverage values above this percentage be included in the output. Optional. <percent> is the coverage of coverpoints and crosses, not covergroups.
- **-all**
When reporting toggles, creates a report that lists both toggled and untoggled signals. Counts of all enumeration values are reported. Not a valid option when reporting on a functional coverage database. Optional.
- **-append**
Appends the report data to the named output file. Optional.
- **-assert**
Reports only assertion coverage data. Optional.
- **-below <percent>**
Specifies that only objects with coverage values below this percentage be included in the output. Optional. <percent> is the coverage of coverpoints and crosses, not covergroups. See also **-above**.

- **-binrhs**
Specifies that for covergroups, a column is included in the report that displays the RHS for covergroup bins. The RHS is a sampled value that cause the bin to increment. Optional.
- **-bydu**
Reports coverage statistics by design unit (du). Optional. The simulator will iterate through all design units/modules in the design and report coverage data for each. Each design unit report will be the sum of all instances of that module and will be sorted by design unit name. Can be used with the **-recursive [-depth <n>]** argument to report on all design units contained within the specified design unit. You can also report coverage data for a specific design unit by using the **-du <name>** argument.
- **-byfile**
Writes out a coverage summary for each source file in the design. Optional. This is the default report generated. A report generated with **-byfile** does not contain toggle information.
- **-byinstance**
Writes out a coverage summary for all instances and packages. The default setting, if not used, is **-byfile**. Optional.
- **-code {b | c | e | f | s | t}...**
Specifies which code coverage statistics to include in the report. Optional. If this argument is not specified, the report includes statistics for all categories you enabled at compile time.
Not a valid option for functional coverage data.

The characters are as follows:

- b — Include branch statistics.
- c — Include condition statistics.
- e — Include expression statistics.
- f — Include finite state machine statistics.
- s — Include statement statistics.
- t — Include toggle statistics.

For example, to includes statistics associated with each coverage item except toggles in the report, you would enter “**-code bcefs**”.

Note that extended toggle coverage will be reported only if the design which created the UCDB was compiled (vlog/vcom) with the **-code x** argument, and the **vcover** command was submitted with **-code t**.

- **-concurrent | -immediate**
(optional) Only concurrent assertions are selected when the **-concurrent** switch is used. Only immediate assertions are selected when the **-immediate** switch is used. These are mutually

exclusive options. If neither switch is specified, both concurrent and immediate assertions are selected.

- -covered

Specifies that only the covered bins of a coverpoint or cross appear in the report. Optional. This argument applies to covergroups only. If any covergroup item survives the covergroup filters applied, its ancestors are displayed in the report and the GUI.

- -coverenhanced

Enables non-critical functionality which might change the appearance or content of coverage metrics. This argument only has an effect in letter releases (6.6a, 6.6b, etc.) . In major releases (6.6, 6.7, etc.), all coverage enhancements present in previous letter release streams are enabled by default, and -coverenhanced is no longer necessary to enable these enhancements. Bug fixes important to the correctness of coverage numbers are always enabled by default, with no need for -coverenhanced. Since the exact nature of -coverenhanced varies from release to release, the details of the enhancements it enables are present in the product release notes rather than in the Command Reference. For these details, search the release notes using the string "coverenhanced".

- -cvg

Reports only covergroup coverage data. Optional.

- -details [-dumptables] [-fecanalysis] [-metricanalysis]

Includes details associated with each coverage item in the output (both UDP and FEC). By default, details are not provided. Optional.

-dumptables — forces printing of condition and expression truth tables even though fully covered. Optional.

-fecanalysis — reports which input patterns can be applied to the inputs to increment the expression/condition hit counts. Optional.

-metricanalysis — prints sum-of-product and basic sub-condition heuristic metrics from UDP expression/condition view. It reports hit counts for all rows in UPD table. To improve coverage numbers, find rows with 0 hits and exercise the inputs accordingly. See “[Condition and Expression Coverage](#)” for more information on metrics. Optional.

- -directive

Reports only directives coverage data. Optional.

- -du <du_name>

Reports coverage statistics for the specified design unit. Optional. <du_name> is <library name>.<primary>(<secondary>), where the library name is optional, and secondary name is required only for VHDL. If there are parameterized instances, all are considered to match the specified design unit.

- **-duplicates**
Valid with **-toggle** switch. Reports all toggle nodes, including alias nodes, using their local names. Optional.
- **-error <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "error." Optional. Does not function with internal messages (those without numbers). Edit the **error** variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-excluded [-pragma | -user]**
Includes details on the exclusions in the specified coverage database input file. Optional. By default, this option includes both user exclusions and source code pragma exclusions, unless you specify **-user** or **-pragma**. The output is structured in DO file command format.
 - pragma — When used with the **-excluded** argument, writes out *only* lines currently being excluded by pragmas. Optional.
 - user — When used with the **-excluded** argument, writes out files and lines currently being excluded by the **coverage exclude** command. Optional.
- **-file <filename>**
Specifies a file name for the report. Optional. Default is to write the report to the Transcript window. Environment variables may be used in the pathname.
- **-html [-code [bcesf[t|x]] [-assert] [-cvg] [-directive] [-verbose] [-nosource] [-noframes] [-nodetails] [-summary] [-htmldir <outdir>] [-threshL <val>] [-threshH <val>] <input_ucdb>**
Generate an HTML coverage report on coverage data from a given UCDB file. Optional. You can use the **-verbose** option with **-html** to enable logging output for each file generated. The **-html** arguments listed below are not compatible with any other vcover report arguments, with the exception of **-binrhs**, **-code**, **-assert**, **-cvg**, and **-directive**.
 - <input_ucdb> — Specifies input UCDB file. Required, and only one is allowed.
 - [-code [bcesf[t|x]] [-assert] [-cvg] [-directive] — Specifies types of coverage to include in the HTML coverage report. If none of these four switches is defined, all coverage types are included in the report. A **-covergroup** switch exists which is synonymous to **-cvg** and may be used in its place.
 - verbose — Prints out the files that are generated by the HTML report generator. Optional.
 - nosource — Used to avoid generation of the annotated source. Optional. This argument used if you have no source code, or if you don't want the annotated source to be generated. Note that this prevents you from accessing source code related data from inside the generated HTML report.
 - noframes — Avoids generation of JavaScript-based tree for designs with a large number of design scopes. The report comes up as a single frame containing the top-

level summary page and an HTML-only design scope index page is available as a link from the top-level page.

- nodetails — Omits coverage detail pages, saving time and disk space during report generation for very large designs.
- summary — Includes only the top summary page, the testplan summary page, and the list of tests run in the generated report.
- htmldir <outdir> — Specifies the name of output directory for resulting UCDB (default: "covhtmlreport"). Optional. Whether you specify an output directory or the default is used, any file or directory of that name is completely removed prior to report generation to prevent possible stale data.
- threshL <%> -threshH <val> — Specifies % of coverage at which colored cells change from red to yellow. Optional.
- threshH <%> — Specifies % of coverage at which colored cells change from yellow to green. Optional.

The default output filename is *index.html* in the default directory, *covhtmlreport*.

- -instance <path>
Writes out the source file summary coverage data for the selected instance. Optional.
- -lang sva | psl | vhdl
(optional) Specifies assertions of a specific language (SVA, PSL, or VHDL). You may specify multiple languages by using this option multiple times. If -lang is not specified, all three languages are selected.
- -library <libname>
Only needs to be used when you have packages of the same name in different libraries. Optional.
- -memory
Reports a coarse-grain analysis of capacity data for the following SystemVerilog constructs:
 - Classes
 - Queues, dynamic arrays, and associative arrays (QDAS)
 - Assertion and cover directives
 - Covergroups
 - Solver (calls to randomize())

Optional. When combined with -cvg and -details, this command reports the detailed memory usage of covergroup. These include the current persistent memory, current transient memory, peak transient memory, and peak time of the following:

- Per covergroup type
- Per coverpoint and cross in the type

- Per covergroup instance (if applicable)
- Per coverpoint and cross in the instance (if applicable).
- **-modelsimini <ini_filepath>**

Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-noannotate**

Removes source code from the output report. Valid for code coverage only. Not applicable with **-xml** argument. Optional.
- **-nodetails**

Excludes details associated with each coverage item from the output. Details are included by default. Optional.
- **-noexcludedhits**

By default, if any excluded items are hit during simulation an “E-hit” notification is displayed in the text report. If “E-hit” is present, it overrides the display of “E” for an exclusion or “EA” for an autoexclusion. To disable display of “E-hit” notifications, use the **-noexcludedhits** option. This has the effect of making all “E” and “EA” notifications uniformly visible, along with exclusion reasons (if available). Optional.
- **-noignorebin**

Removes covergroup ignore bins from the report output. Optional.
- **-nomissing**

Removes the Misses column from the report output. Optional.
- **-nozeroweights**

Removes covergroups, coverpoints, and crosses having zero weight from the report output.
- **-note <msg_number>[,<msg_number>,...]**

Changes the severity level of the specified message(s) to "note." Optional. Does not function with internal messages (those without numbers). Edit the **note** variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-option**

Includes all covergroup **option** and **type_option** values in the report. Unless your covergroup has the "option.per_instance" set to true, only the **type_option** is included by default. Only applicable to covergroup reports created with the **-details** argument. Optional.
- **-package <pkgname>**

Prints a report on the specified VHDL package body. Needs to be of the form *<lib>.<pkg>*. Optional. This argument is equivalent to **-du**.

Not a valid option when reporting on a functional coverage database.

- -precision <int>
Sets the decimal precision for printing functional coverage information. Valid values for <int> are from 0 to 6 and default value is 1 (one). Optional.
- -recursive [-depth <n>]
Reports on the instance specified with **-instance** and every included instance, recursively. Can also be used with **-details** and **-totals**. Optional.
 - depth <n>
Used with the **-recursive** argument, it specifies the maximum recursive depth. A depth of 1 is the same as no recursion at all. Optional.
- -samples <spec>
Filters the sample count for covergroup types. The sample count is optionally calculated using [SVCovergroupSampleInfo](#), set in the *modelsim.ini* file. This argument filters to include covergroups whose sample count matches the criteria given <spec>. It is a filter for covergroup types only, not instances. If covergroups are found without a sample count, a warning is issued. <spec> is one of the following four mutually exclusive options:
 - [number] — sample count is equal to [number]
 - le [number] — sample count is less than or equal to number
 - ge [number] — sample count is greater than or equal to number
 - [lower]-[upper] — (no white space allowed) sample count is in the range lower-upper. There is only 1 -sample option allowed per invocation.
- -select {inputs | outputs | inout | ports | internals}
Reports on input, output, inout, all ports, or internal signals. Can be used with the -toggles argument. Optional.
- -severity info | note | warning | error | failure | fatal
(optional) Specifies the assertion severity level. When -severity is specified only assertions with the same or higher severity are selected. If not specified, assertions of all severities are selected.
- -showambiguity
When used, coverage report displays both minimum and maximum counts for any conflicting toggle data in a UCDB that results from a combined merge ([vcover merge](#) command performed with -combine).
- -source <filename>
Writes a summary of statement coverage data for a specific source file. Optional. Environment variables may be used in the pathname.

- **-suppress <msg_number>[,<msg_number>,...]**
Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress either Fatal or Internal messages (without numbers). Edit the [suppress](#) variable in the *modelsim.ini* file to set a permanent default.
- **-testattr**
Display test attributes in the report. Optional.
- **-testextract <test_name_or_pattern>**
Display test specific results in the report. Optional. Used to combine results from multiple tests. The <test_name_or_pattern> is the test or pattern to extract. Multiple -testextract arguments can be applied in same command. This argument is compatible with reports generated in plain text and XML formats only, HTML reports are not supported. When using this argument, a header line appears at the top of the report listing test name(s) used to generate the report. Also, the word “hit” appears in place of the count number. UCDB files store only the aggregated coverage counts from all tests, and test-specific numbers can’t be reproduced.
- **-totals**
Writes out a total summary of the specified instance, recursively. Optional. Useful for tracking changes. Without this argument, the report writes out an instance summary for each of the instances. The report prints only one summary if -totals option is used. Also, when the -totals argument is specified, the alias nodes are not counted.

Not a valid option when reporting on a functional coverage database.
- **-toggles**
Writes out all (and only) the toggles in the entire design (not including alias nodes), or under the instance specified by -instance <path>. Optional. Valid during simulation, post-processing, and in vcover. The toggle report generated with this argument is written in the style of reports generated by [toggle report](#).
- **-top**
For signals that were added to toggle coverage using vcom or vlog -cover t, -top uses the name of the top-most element of multiple-segment (collapsed) nets. Optional. By default the name of the wildcard-matching segment will be used.
- **-unattempteddimmed**
(optional) Causes the unexecuted (unattempted) immediate assertions to be considered in Total Coverage calculations that are displayed in the coverage report. By default, any unexecuted immediate assertions are not included in the coverage calculations displayed in the coverage report.
- **-usecnpm**
Specifies that the value of [SVCrossNumPrintMissingDefault](#) is used in report. By default, all cross bins are displayed in the report.

- -verbose
Prints a report listing all the integer values and their counts an integer toggle encounters during the run. Optional. List will include the number of active assertion threads (Active Count) and number of active root threads (Peak Active Count) that have occurred up to the current time.
- -version
Returns the version number of UCDB file used to create the report. This argument can not be combined with any other arguments; when present, it invalidates all other arguments. Optional.
- -warning <msg_number>[,<msg_number>,...]
Changes the severity level of the specified message(s) to "warning." Optional. Does not function with internal messages (those without numbers). Edit the [warning](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- -xml
Outputs report in XML format. A report created with -xml does not contain source file lines (calls **-noannotate** implicitly). Optional. This implicitly sets the **-details** argument. Refer to “[Coverage Reports](#)” for more information.
- -zeros
Writes out a file-based summary of lines, including file names and line numbers, that have not been executed (zero hits), annotates the source code, and supports the **-source** and **-instance** options. Optional. For covergroups, this argument applies to coverpoint and cross bins only. Optional. Cannot be used in tandem with the **-recursive** argument.

For a detailed report that includes line numbers, use: **vcover report -zeros -details**.
- <file>
Specifies the previously saved code or functional coverage file on which you want to report. Required.

Examples

- Write a top-level summary of the number of instances, statements, branches, hits, and signal toggles to *myreport.txt*.

```
vcover report -totals -file myreport.txt input.ucdb
```
- Write detailed branch, condition, and statement statistics from *save.ucdb*, without associated source code, to stdout.

```
vcover report -details -code bcs save.ucdb
```
- Write a summary of code coverage for all instances in *save.cov* to stdout.

```
vcover report save.ucdb
```

- Write code coverage details of all instances in *input.ucdb* to *save.cov*. The **-details** option reports coverage statistics for each statement and branch. Branch coverage statistics will follow statement statistics and will be presented in four columns: line, column, true branch count, false branch count.

```
vcover report -details -file save.cov input.ucdb
```

- Write code coverage details of one specific instance to *save.cov*.

```
vcover report -details -instance /top/p -file save.cov input.ucdb
```

- Write a summary of coverage by source file for coverage less than or equal to 90%.

```
vcover report -details -below 90 -file myreport.txt input.ucdb
```

- Write a list of statements with zero coverage to *myzerocov.txt*.

```
vcover report -zeros -file myzerocov.txt input.ucdb
```

See also

[“Code Coverage”](#), [coverage save](#), [“Verification with Assertions and Cover Directives”](#), [“Verification with Functional Coverage”](#), [coverage report](#), [coverage ranktest](#)

vcover stats

The **vcover stats** command computes and prints to *stdout* summary statistics for previously saved code or functional coverage databases. It can be invoked within the ModelSim GUI or at the command line.

vcover stats creates coverage statistics output that is equivalent to the output from this command:

```
vcover report -totals -byinstance
```

Syntax

```
vcover stats [-assert] [-code {b | c | e | f | s | t}...] [-cvg] [-directive] [-error  
  <msg_number>[,<msg_number>,...]] [-inputs <pathname>]  
  [-memory] [-modelsimini <ini_filepath>] [-precision <int>]  
  [-error <msg_number>[,<msg_number>,...]]  
  [-note <msg_number>[,<msg_number>,...]]  
  [-suppress <msg_number>[,<msg_number>,...]]  
  [-warning <msg_number>[,<msg_number>,...]]  
  <file1> [<file2> <file>...]
```

Arguments

- **-assert**
Reports only assertion coverage data. Optional. Includes Assertion Passes, Failures, Successes and Attempts. At simulation, time if "-assertdebug" is present, Assertion Passes are counted. Assertion Successes are those assertions that never failed and passed at least once. In the absence of "-assertdebug," Assertion Passes are not counted, and Assertion Successes are those assertions that never failed. Assertion Attempts are the sum of all evaluated assertions.
- **-code {b | c | e | f | s | t}...**
Specifies which code coverage statistics to include in the report. Optional. By default the report includes statistics for all categories you enabled at compile time.

This argument is ignored when the UCDB does not contain any code coverage data.

The characters are as follows:

- b — Include branch statistics.
- c — Include condition statistics.
- e — Include expression statistics.
- f — Include finite state machine statistics.
- s — Include statement statistics.
- t — Include toggle statistics.

- **-cvg**
Reports only covergroup coverage data. Optional.
- **-directive**
Reports only directives coverage data. Optional.
- **-error <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "error." Optional. Does not function with internal messages (those without numbers). Edit the [error](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **<file1> [<file2> <filen>...]**
Specifies the file(s) for which you want summary statistics, including *.rank* file created by [coverage ranktest](#) or [vcover ranktest](#). Required. Multiple pathnames and wildcards are allowed.
- **-inputs <pathname>**
Specifies a text file containing input filenames for which you want to produce statistics. Optional.
- **-memory**
Reports a coarse-grain analysis of capacity data for the following SystemVerilog constructs (Optional):
 - Classes
 - Queues, dynamic arrays, and associative arrays (QDAS)
 - Assertion and cover directives
 - Covergroups
 - Solver (calls to `randomize()`)
- **-modelsimini <ini_filepath>**
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-note <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "note." Optional. Does not function with internal messages (those without numbers). Edit the [note](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

- `-precision <int>`

Sets the decimal precision for printing coverage information. Valid values for <int> are from 0 to 6 and default value is 1 (one). Optional. Only the output display is affected by this argument, NOT the contents of the UCDB itself.
- `-suppress <msg_number>[,<msg_number>,...]`

Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress either Fatal or Internal messages (without numbers). Edit the [suppress](#) variable in the *modelsim.ini* file to set a permanent default.
- `-warning <msg_number>[,<msg_number>,...]`

Changes the severity level of the specified message(s) to "warning." Optional. Does not function with internal messages (those without numbers). Edit the [warning](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

See also

[coverage save](#), [vcover merge](#), “[Code Coverage](#)”, [coverage report](#), “[Verification with Functional Coverage](#)”, “[Verification with Assertions and Cover Directives](#)”, [vcover ranktest](#), [coverage ranktest](#)

vcover testnames

The **vcover testnames** command displays the testnames in the currently loaded UCDB file. If a merged file, it gives you a list of tests in the merged file.

This command is most useful if you use the `-testextract` from a [coverage analyze](#) or [vcover report](#) command, because it requires the test name. By default, the testname is the name of the UCDB file, though you can set it to whatever you want.

Syntax

```
coverage testnames [-tcl] [-modelsimini <ini_filepath>] [-error  
  <msg_number>[,<msg_number>,...]] [-note <msg_number>[,<msg_number>,...]] [-  
  suppress <msg_number>[,<msg_number>,...]] [-warning  
  <msg_number>[,<msg_number>,...]]
```

Arguments

- `-error <msg_number>[,<msg_number>,...]`
Changes the severity level of the specified message(s) to "error." Optional. Does not function with internal messages (those without numbers). Edit the `error` variable in the `modelsim.ini` file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- `-modelsimini <ini_filepath>`
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- `-note <msg_number>[,<msg_number>,...]`
Changes the severity level of the specified message(s) to "note." Optional. Does not function with internal messages (those without numbers). Edit the `note` variable in the `modelsim.ini` file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- `-suppress <msg_number>[,<msg_number>,...]`
Prevents the specified message(s) from displaying. The `<msg_number>` is the number preceding the message you wish to suppress. Optional. You cannot suppress either Fatal or Internal messages (without numbers). Edit the `suppress` variable in the `modelsim.ini` file to set a permanent default.
- `-tcl`
Print attribute information in a tcl format. Optional.
- `-warning <msg_number>[,<msg_number>,...]`
Changes the severity level of the specified message(s) to "warning." Optional. Does not function with internal messages (those without numbers). Edit the `warning` variable in the

modelsim.ini file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

See also

[Code Coverage](#), [“Verification Browser Window”](#), [“Verification Tracker Window”](#), [coverage analyze](#), [coverage attribute](#), [coverage exclude](#), [coverage goal](#), [coverage ranktest](#), [coverage report](#), [coverage save](#), [coverage tag](#), [coverage testnames](#), [coverage weight](#), [vcover merge](#), [vcover ranktest](#), [vcover stats](#)

vdbg

This command performs a pre-simulation debug analysis of the sequential and combinatorial elements in your design. The results are added to the optimized design unit and picked up when a simulation is run for later causality tracing and schematic functional representation.

Note



To perform a causality trace, logging of simulation data must be enabled with the [add wave](#) command or the [log](#) command.

Prerequisites

The design must be optimized with the **vopt** +acc command before executing the **vdbg** command.

Syntax

```
vdbg [-work <library_name>] <opt_dut_name>
```

Arguments

- **-work <library_name>**
(optional) Specifies the location of the optimized design unit.
<library_name> — The name of the design library containing **<opt_dut_name>**.
- **-Wvopt <value>**
Passes additional arguments to vopt.
<value> — One or more arguments to the [vopt](#) command. Multiple arguments must be separate by a space and enclosed in quotation marks (“ ”).
- **opt_dut_name**
(required) The name of the optimized design unit specified with vopt.

Examples

- The following is a four-step workflow that sets up the conditions for debugging a design with Causality Traceback.

```
vlog foo.v  
vopt foo -o foo_dbg +acc  
vdbg foo_dbg  
vsim foo_dbg -debugdb
```

Related Topics

- [add wave](#)
- [find drivers](#)
- [log](#)
- [vsim -debugDB](#)
- [Using Causality Traceback](#)

vdel

This command deletes a design unit from a specified library.

This command provides additional information with the **-help** switch.

Syntax

```
vdel [-lib <library_path>] [-modelsimini <ini_filepath>] [-verbose]
      [-all | <primary> [<arch_name>] | -allsystemc] [-obj {<object> | all}]
```

Arguments

- **-all**
(optional) Deletes an entire library.

Caution



BE CAREFUL! Libraries cannot be recovered once deleted. You are not prompted for confirmation.

- **-allsystemc**
(optional) Deletes all SystemC modules in a design from the working directory. Optional.
- **<arch_name>**
(optional) Specifies the name of an architecture to be deleted. Optional. If omitted, all of the architectures for the specified entity are deleted. Invalid for a configuration or a package.
- **<primary>**
(required unless **-all** is used) Specifies the entity, package, configuration, or module to be deleted. This option is not supported for SystemC modules.
- **-lib <library_path>**
(optional) Specifies location of the library that holds the design unit to be deleted. By default, the design unit is deleted from the **work** library.
<library_path> — A logical name or pathname of the library.
- **-modelsimini <ini_filepath>**
(optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable.
<ini_filepath> — Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-obj {<object> | all}**
(optional) removes directories containing SystemC and DPI object files.
<object> — Specifies the type of directory to remove, as reported by the output of the **vdir -obj** command. This will take the form of either:
<compiler> — a string identifying the compiler, such as **gcc-3.3.1**.

<platform> — a string identifying the platform, such as **linux** or **sunos5**.

<platform-compiler> — a string identifying a compiler/platform pair, such as **linux_gcc-3.2.3** or **sunos5_gcc-4.1.1**.

all — Specifies that all directories should be removed, as reported by the output of the **vdir -obj** command.

- **-verbose**
(optional) Displays progress messages.

Examples

- Delete the **work** library.
vdel -all
- Delete the **synopsys** library.
vdel -lib synopsys -all
- Delete the entity named **xor** and all its architectures from the **work** library.
vdel xor
- Delete the architecture named **behavior** of the entity **xor** from the **work** library.
vdel xor behavior
- Delete the package named **base** from the **work** library.
vdel base

vdir

This command lists the contents of a design library.

This command also checks the compatibility of a vendor library. If **vdir** cannot read a vendor-supplied library, the library may not be compatible with ModelSim.

This command lists SystemC modules that are exported with the `SC_MODULE_EXPORT()` macro.

This command provides additional information with the **-help** switch.

Syntax

```
vdir [-l | [-prop <prop>]] [-r] [-obj] [-all | [-lib <library_name>]] [<design_unit>]]  
    [-modelsimini <ini_filepath>]
```

Arguments

- **-all**
(optional) Lists the contents of all libraries listed in the Library section of the active *modelsim.ini* file. Refer to [modelsim.ini Variables](#) for more information.
- **<design_unit>**
(optional) Indicates the design unit to search for within the specified library. If the design unit is a VHDL entity, its architectures are listed. By default all entities, configurations, modules, packages, and optimized design units in the specified library are listed.
- **-l**
(optional) Prints the version of **vcom/vlog/scom** with which each design unit was compiled, plus any compilation options used. Also prints the object-code version number that indicates which versions of **vcom/vlog/scom** and ModelSim are compatible.
- **-lib <library_name>**
(optional) Specifies the logical name or the pathname of a library to be listed. By default, the contents of the **work** library are listed.
<library_name> — A logical name or pathname of a library.
- **-modelsimini <ini_filepath>**
(optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable.
<ini_filepath> — Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-obj**
(optional) *SystemC only*. Returns the content of the object directories, such as *<work>_sc* or *<work>/_dpi*.

- -prop <prop>

(optional) Reports on a specified design unit property.

<prop> — Specifies a Design Unit Property, as listed in [Table 2-10](#). If you do not specify a value for <prop>, an error message is displayed.

Table 2-10. Design Unit Properties

Value of <prop>	Description
archcfg	configuration for arch
bbox	blackbox for optimized design
body	needs a body
default	default options
dir	source directory
dpnd	depends on
entcfg	configuration for entity
extern	package reference number
inline	module inlined
lock	lock/unlock status
lrm	language standard
mtime	source modified time
name	short name
opcode	opcode format
options	compile options
root	optimized Verilog design root
src	source file
top	top level model
ver	version string
vlogv	Verilog version
voptv	Verilog optimized version

- -r

(optional) Prints architecture information for each entity in the output.

Examples

- List the architectures associated with the entity named **my_asic** that reside in the HDL design library called **design**.

vdir -lib -r design my_asic

- Show the output of **vdir -l**, including any compilation options used to compile the library:

```
> # MODULE ram_tb
> #   Verilog Version: RV9il?9FGhibjG<jXXV_`1
> #   Version number: CRW2<UhheaW;LIL2_B5o31
> #   Source modified time: 1132284874
> #   Source file: ram_tb.v
> #   Start source location: ram_tb.v:47
> #   Version number: CRW2<UhheaW;LIL2_B5o31
> #   Opcode format: 6.1c; VLOG SE Object version 31
> #   Optimized Verilog design root: 1
> #   Language standard: 1
> #   Compile options: -cover bcst
> #   Compile defaults: GenerateLoopIterationMax=100000
> #   Source directory: C:\Verif\QuestaSim_6.1c
                        \examples\tutorials\verilog\memory
```

vencrypt

This command encrypts Verilog and SystemVerilog code contained within encryption envelopes. The code is not pre-processed before encryption, so macros and other `directives are unchanged. This allows IP vendors to deliver encrypted IP with undefined macros and `directives.

Upon execution of this command, the filename extension will be changed to *.vp* for Verilog files (*.v* files) and *.svp* for SystemVerilog files (*.sv* files).

If the `vencrypt` utility processes the file (or files) and does not find any encryption directives it reprocesses the file using the following default encryption:

```
`pragma protect data_method = "aes128-cbc"  
`pragma protect key_keyowner = "MTI"  
`pragma protect key_keyname = "MGC-DVT-MTI"  
`pragma protect key_method = "rsa"  
`pragma protect key_block encoding = (enctype = "base64")  
`pragma protect begin
```

The `vencrypt` command must be followed by a compile command – such as `vlog` – for the design to be compiled.

This command provides additional information with the **-help** switch.

Syntax

```
vencrypt <filename> [-d <dirname>] [-e <extension>] [-f <filename>] [-h <filename>]  
[-l <filename>] [-o <filename>] [-p <prefix>] [-quiet]
```

- <filename>
(required) Specifies the name of the Verilog source code file to encrypt. One filename is required. Multiple filenames can be entered separated by spaces. Wildcards can be used. Default encryption pragmas will be used, as described above, if no encryption directives are found during processing.
- -d <dirname>
(optional) Specifies where to save encrypted Verilog files. If no directory is specified, current working directory will be used.

 <dirname> — Specifies the directory to contain the encrypted Verilog or System Verilog files. The original file extension (*.v* for Verilog and *.sv* for SystemVerilog) will be preserved.
- -e <extension>
(optional) Specifies a filename extension.

 <extension> — Any alpha-numeric string.
- -f <filename>
(optional) Specifies a file with more command line arguments. Allows complex arguments to be reused without retyping. Nesting of **-f** options is allowed.

Refer to the section "[Argument Files](#)" for more information.

<filename> — Specifies the name of a file containing command line arguments.

- -h <filename>

(optional) Concatenates header information into all design files listed with <filename>. Allows the user to pass a large number of files to the vencrypt utility that do not contain the ``pragma protect` or ``protect` information about how to encrypt the file. Saves the user from editing hundreds of files to add in the same ``pragma protect` to every file.

<filename> — Specifies an existing file.

- -l <filename>

(optional) Redirects log output to the file designated by <filename>.

<filename> — Specifies a file for saving output.

- -o <filename>

(optional) Combines all encrypted output into a single file.

<filename> — Specifies a file for saving output.

- -p <prefix>

(optional) Prepends file names with a prefix.

<prefix> — Any alpha-numeric string.

- -quiet

(optional) Disables encryption messages.

Example

- Insert header information into all design files listed.

```
vencrypt -h encrypt_head top.v cache.v gates.v memory.v
```

The `encrypt_head` file may look like the following:

```
`pragma protect data_method = "aes128-cbc"  
`pragma protect author = "IP Provider"  
`pragma protect key_keyowner = "MTI", key_method = "rsa"  
`pragma protect key_keyname = "MGC-DVT-MTI"  
`pragma protect begin
```

There is no ``pragma protect end` expression in the header file, just the header block that starts the encryption. The ``pragma protect end` expression is implied by the end of the file. For more detailed examples, see "[Protecting Your Source Code](#)" in the User's Manual.

Related Topics

- [vhencrypt](#)
- "[Protecting Your Source Code](#)" in the User's Manual

verror

This command prints a detailed description about a message number. It may also point to additional documentation related to the error.

This command provides additional information with the **-help** switch.

Syntax

verror [-fmt | -tag | -fmt -tag | -full] <msgNum> ...

verror [-fmt | -tag | -fmt -tag | -full] [-tool <tool>] -all

verror [-tool <tool>] -ranges

Arguments

- -fmt | -tag | -fmt -tag | -full

(optional) Specifies the type and amount of information to return.

-fmt — returns the format string used in the error message.

-tag — returns the tag associated with the error message.

-full — returns the format string, tag, and complete text associated with the error message.

- [-tool <tool>] -all

(required when not specifying <msgNum> or **-ranges**) Returns information about all messages associated with a specified tool, where <tool> can be one of the following:

common	vcom	vcom-vlog
vlog	vsim	vsim-vish
wlf	vsim-sccom	sccom
vsim-systemc	ucdb	vsim-vlog
pseudo_synth		

- <msgNum>

(required when not specifying **-all** or **-ranges**) Specifies the message number(s) you would like more information about. You can find the message number in messages of the format:

** <Level>: ([<Tool>-[<Group>-])<MsgNum>) <FormattedMsg>

You can specify <msgNum> any number of times for one verror command in a space-separated list.

- -ranges

(required when not specifying <msgNum> or **-all**) Prints the numeric ranges of error message numbers, organized by tool.

Example

- If you receive the following message in the transcript:

```
** Error (vsim-3061) foo.v(22): Too many Verilog port connections.
```

and you would like more information about this message, you would type:

```
verror 3061
```

and receive the following output:

```
Message # 3061:  
Too many Verilog ports were specified in a mixed VHDL/Verilog  
instantiation. Verify that the correct VHDL/Verilog connection is  
being made and that the number of ports matches.  
[DOC: ModelSim User's Manual - Mixed VHDL and Verilog Designs  
Chapter]
```

vgencomp

Once a Verilog module is compiled into a library, you can use this command to write its equivalent VHDL component declaration to standard output.

Optional switches allow you to generate bit or vl_logic port types; std_logic port types are generated by default.

This command provides additional information with the **-help** switch.

Syntax

```
vgencomp [-lib <library_name>] [-b] [-modelsimini <ini_filepath>] [-s] [-v] <module_name>
```

Arguments

- **-lib <library_name>**
(optional) Specifies the working library where the default is to use the **work** library.
<library_name> — Specifies the path and name of the working library.
- **-b**
(optional) Causes **vgencomp** to generate bit port types.
- **-modelsimini <ini_filepath>**
(optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable.
<ini_filepath> — Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-s**
(optional) Used for the explicit declaration of default std_logic port types.
- **-v**
(optional) Causes **vgencomp** to generate vl_logic port types.
- **<module_name>**
(required) Specifies the name of the Verilog module to be accessed.

Examples

- This example uses a Verilog module that is compiled into the **work** library. The module begins as Verilog source code:

```
module top(i1, o1, o2, io1);
  parameter width = 8;
  parameter delay = 4.5;
  parameter filename = "file.in";

  input i1;
  output [7:0] o1;
  output [4:7] o2;
  inout [width-1:0] io1;
endmodule
```

After compiling, **vgencomp** is invoked on the compiled module:

vgencomp top

and writes the following to stdout:

```
component top
  generic(
    width          : integer := 8;
    delay         : real    := 4.500000;
    filename      : string  := "file.in"
  );
  port(
    i1            : in     std_logic;
    o1            : out    std_logic_vector(7 downto 0);
    o2            : out    std_logic_vector(4 to 7);
    io1          : inout  std_logic_vector
  );
end component;
```

vhencrypt

This command encrypts VHDL code contained within encryption envelopes. The code is not compiled before encryption, so dependent packages and design units do not have to exist before encryption.

Upon execution of this command, the *.vhd* filename extension is changed to *.vhdp* and the *.vhdl* filename extension is changed to *.vhdlp*.

If the `vhencrypt` utility does not find any encryption directives, no output file is produced.

The `vhencrypt` command must be followed by a compile command – such as `vcom` – for the design to be compiled.

This command provides additional information with the **-help** switch.

Syntax

```
vhencrypt <filename> [-d <dirname>] [-e <extension>] [-f <filename>] [-h <filename>]  
[-l <filename>] [-o <filename>] [-p <prefix>] [-quiet]
```

- <filename>
(required) Specifies the name of the VHDL source code file to encrypt. One filename is required. Multiple filenames can be entered separated by spaces. Wildcards can be used.
- -d <dirname>
(optional) Specifies where to save encrypted VHDL files. If no directory is specified, current working directory will be used.

 <dirname> — Specifies the directory to contain the encrypted VHDL files. The original file extension (*.vhd* or *.vhdl*) will be preserved.
- -e <extension>
(optional) Specifies a filename extension to be applied to the encrypted file.

 <extension> — Any alpha-numeric string.
- -f <filename>
(optional) Specifies a file with more command line arguments. Allows complex arguments to be reused without retyping. Nesting of **-f** options is allowed.

 Refer to the section "[Argument Files](#)" for more information.

 <filename> — Specifies the name of a file containing command line arguments.
- -h <filename>
(optional) Concatenates header information into all design files listed with <filename>. Allows the user to pass a large number of files to the `vhencrypt` utility that do not contain the encryption information (between the ``protect` and ``protect end` directives) about how to encrypt the file. Saves the user from editing hundreds of files to add the same encryption information into every file.

 <filename> — Specifies an existing file.

- **-l <filename>**
(optional) Redirects log output to the file designated by <filename>.
 <filename> — Specifies a file for saving output.
- **-o <filename>**
(optional) Combines all encrypted output into a single file.
 <filename> — Specifies a file for saving output.
- **-p <prefix>**
(optional) Prepends encrypted file names with a prefix.
 <prefix> — Any alpha-numeric string.
- **-quiet**
(optional) Disables encryption messages.

Related Topics

- [vencrypt](#)
- "[Protecting Your Source Code](#)" in the User's Manual

view

This command opens the specified window. Without arguments this command returns a list of windows currently being viewed.

When you use the **-new** argument with the **view** command ModelSim will create an additional instance of the specified window type and make it the active window for that type. If multiple instances of a window exist, **view** will change the active window of that type to the specified window.

Names for windows are generated as follows:

- The first window name (automatically generated without using **-new**) has the same name as the specified window type. For example, the **view wave** command will create a wave window named "wave."
- Additional window names, created by using the **-new** argument, appends an integer to the window type, starting with 1. For example, the next time you use the **view wave** command, the automatically generated window name will be "wave1." Use the command again and the name will be "wave2," then "wave3," etc.
- You can rename existing windows with the **-title** argument or create a name for a new window using **-new** and **-title** together.

To remove a window, use the **noview** command.

The **view** command with one or more options and no window names specified applies the options to the currently open windows. See examples for additional details.

Syntax

```
view <window_type>... [-aliases] [-names] [-new] [-title {New Window Title}]  
[-undock {[-icon] [-height <n>] [-width <n>] [-x <n>] [-y <n>]}] | -dock
```

Arguments

- <window_type>...
(required) Specifies the window type to view. You do not need to type the full type name (see examples below); implicit wildcards are accepted; multiple window types are accepted. Available window types are:

assertions	atv	branch	browser
calltree	capacity	classgraph	classtree
condition	covergroups	details	duranked
exclusions	expression	fcovers	files
fsmcoverage	fsmlist	fsmview	instance
library	list	locals	memdata
memory	msgviewer	objects	process

profiledetails	project	ranked	runmgr
source	stackview	statement	structural
structure	toggle	tracker	transaction
transcript	watch	wave	dataflow
schematic			

Not all windows are available with all variants (ModelSim SE, ModelSim PE, Questa SV/AFV, etc.)

When you specify a window type and also use the **-new** argument, you create a new instance of that window type. You may also specify the window(s) to view when multiple instances of that window type exist (such as `wave2`). This works only with window names automatically generated by ModelSim, not with window titles specified with the **-title** argument.

- **-aliases**
(optional) Returns a list of `<window_type>` aliases.
- **-height <n>**
(optional) Specifies the window height in pixels. Can only be used with the **-undock** switch.
`<n>` — Any non-negative integer.
- **-icon**
(optional) Toggles the view between window and icon. Can only be used with the **-undock** switch.
- **-names**
(optional) Returns a list of valid `<window_type>` arguments.
- **-new**
(optional) Creates a new instance of the window type specified with the **<window_type>** argument. New window names are automatically generated by appending an integer to the window type, starting with 1, then incrementing the integer when windows of the same type are created.
- **-title {New Window Title}**
(optional) Specifies the window title of the designated window. Curly braces are only needed for titles that include spaces. Double quotes can be used in place of braces, for example "New Window Title". If the new window title does not include spaces, no braces or quotes are needed. For example: *-title new_wave wave* assigns the title *new_wave* to the Wave window.
`{New Window Title}` —
- **-dock**
(optional) Docks the specified standalone window into the Main window.

- **-undock**
(optional) Opens the specified window as a standalone window, undocked from the Main window.
- **-width <n>**
(optional) Specifies the window width in pixels. Can only be used with the **-undock** switch.
<n> — Any non-negative integer.
- **-x <n>**
(optional) Specifies the window upper-left-hand x-coordinate in pixels. Can only be used with the **-undock** switch.
<n> — Any non-negative integer.
- **-y <n>**
(optional) Specifies the window upper-left-hand y-coordinate in pixels. Can only be used with the **-undock** switch.
<n> — Any non-negative integer.

Examples

- Undock the Wave window from the Main window and makes it a standalone window.

```
view -undock wave
```

- Display an undocked Processes window in the upper left-hand corner of the monitor with a window size of 300 pixels, square.

```
view process -undock -x 0 -y 0 -width 300 -height 300
```

- Display the Watch and Wave windows.

```
view w
```

- Display the Objects and Processes windows.

```
view ob pr
```

- Open a new Wave window with My Wave Window as its title.

```
view -title {My Wave Window} wave
```

- The first command creates a window named 'wave'. The second command creates a window named 'wave1'. Its full Tk path is '.wave1'. Wave1 is now the active Wave window. Any **add wave** command would add objects to wave1.

```
view wave  
view wave -new
```

- Change the default Wave window back to 'wave'.

```
view wave
```

- Will override the default Wave window and add *mysig* to wave1.

```
add wave -win .wave1 mysig
```

- Open a new Wave window with "SV_Signals" as its title, then add signals to it.

```
set SV_Signals [view wave -new -title SV_Signals]  
add wave -window $SV_Signals /top/mysignals
```

The custom window title "SV_Signals" is saved as a TCL variable, then called using the '\$' prefix.

See also

[noview](#)

virtual count

The **virtual count** command reports the number of currently defined virtuals that were not read in using a macro file.

Syntax

```
virtual count [-kind {implicits | explicits}] [-unsaved]
```

Arguments

- -kind {implicits | explicits}

(optional) Reports only a subset of virtuals.

implicits — virtual signals created internally by the product.

explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

Unique abbreviations are accepted.

- -unsaved

(optional) Reports the count of only those virtuals that have not been saved to a macro file.

See also

[virtual define](#), [virtual save](#), [virtual show](#), “[Virtual Objects](#)”

virtual define

The **virtual define** command prints to the transcript the definition of the virtual signals, functions, or regions in the form of a command that can be used to re-create the object.

Syntax

```
virtual define [-kind <kind>] <pathname>
```

Arguments

- -kind {implicits | explicits}

(optional) Transcripts only a subset of virtuals.

implicits — virtual signals created internally by the tool.

explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

Unique abbreviations are accepted.

- <pathname>

(required) Specifies the path to the virtual(s) for which you want definitions, where wildcards are allowed.

Examples

- Show the definitions of all the virtuals you have explicitly created.

```
virtual define -kind explicits *
```

See also

[virtual describe](#), [virtual show](#), “[Virtual Objects](#)”

virtual delete

The **virtual delete** command removes the matching virtuals.

Syntax

```
virtual delete [-kind <kind>] <pathname>
```

Arguments

- -kind {implicits | explicits}

(optional) Removes only a subset of virtuals.

implicits — virtual signals created internally by the product.

explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

Unique abbreviations are accepted.

- <pathname>

(required) Specifies the path to the virtual(s) you want to delete, where wildcards are allowed.

Examples

- Delete all of the virtuals you have explicitly created.

```
virtual delete -kind explicits *
```

See also

[virtual signal](#), [virtual function](#), [“Virtual Objects”](#)

virtual describe

The **virtual describe** command prints to the transcript a complete description of the data type of one or more virtual signals.

Similar to the existing **describe** command.

Syntax

```
virtual describe [-kind <kind>] <pathname>
```

Arguments

- `-kind {implicits | explicits}`
(optional) Transcripts only a subset of virtuals.
 - `implicits` — virtual signals created internally by the product.
 - `explicits` — virtual signals explicitly created by a user, such as with the virtual signal command.Unique abbreviations are accepted.
- `<pathname>`
(required) Specifies the path to the virtual(s) for which you want descriptions, where wildcards are allowed.

Examples

- Describe the data type of all virtuals you have explicitly created.

```
virtual describe -kind explicits *
```

See also

[virtual define](#), [virtual show](#), “[Virtual Objects](#)”

virtual expand

The **virtual expand** command prints to the transcript a list of all the non-virtual objects contained in the specified virtual signal(s).

You can use this to create a list of arguments for a command that does not accept or understand virtual signals.

Syntax

```
virtual expand [-base] <pathname> ...
```

Arguments

- **-base**
(optional) Outputs the root signal parent in place of a subelement. For example:

```
vcd add [virtual expand -base myVirtualSignal]
```


the resulting command after substitution would be:

```
vcd add signala signalb signalc
```
- **<pathname>**
(required) Specifies the path to the signals and virtual signals to expand, where wildcards are allowed and you can specify any number of paths.

Examples

- Add the elements of a virtual signal to the VCD file.

In the Tcl language, the square brackets specify that the enclosed command should be executed first ("virtual expand ..."), then the result substituted into the surrounding command.

```
vcd add [virtual expand myVirtualSignal]
```

Therefore, if *myVirtualSignal* is a concatenation of *signala*, *signalb.rec1* and *signalc(5 downto 3)*, the resulting command after substitution would be:

```
vcd add signala signalb.rec1 {signalc(5 downto 3)}
```

The slice of *signalc* is enclosed in curly braces, because it contains spaces.

See also

[virtual signal](#), “[Virtual Objects](#)”

virtual function

The **virtual function** command creates a new signal, known only by the GUI (not the kernel), that consists of logical operations on existing signals and simulation time, as described in **<expressionString>**.

It cannot handle bit selects and slices of Verilog registers. Please see *Syntax and Conventions* for more details on syntax.

If the virtual function references more than a single scalar signal, it will display as an expandable object in the Wave and Objects windows. The children correspond to the inputs of the virtual function. This allows the function to be "expanded" in the Wave window to see the values of each of the input waveforms, which could be useful when using virtual functions to compare two signal values.

Virtual functions can also be used to gate the List window display.

Note



The virtual function and virtual signal commands are interchangeable. The product will keep track of whether you've created a signal or a function with the commands and maintain them appropriately. We document both commands because the virtual save, virtual describe, and virtual define commands will reference your virtual objects using the correct command.

Syntax

```
virtual function [-env <path>] [-install <path>] [-delay <time>] {<expressionString>} <name>
```

Arguments

Arguments for **virtual function** are the same as those for **virtual signal**, except for the contents of the expression string.

- **-env <path>**
(optional) Specifies a hierarchical context for the signal names in **<expressionString>** so they don't all have to be full paths.
- **-install <path>**
(optional) Causes the newly-created signal to become a child of the specified region. If **-install** is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in **<expressionString>**. If the expression references more than one WLF file (dataset), the virtual signal will automatically be placed in region `virtualls:/Functions`.
- **-delay <time>**
(optional) Specifies a value by which the virtual function will be delayed. You can use negative values to look forward in time. If units are specified, the **<time>** option must be enclosed in curly braces. See the examples below for more details.

- {<expressionString>}
(required) A text string expression, enclosed in curly braces ({ }) using the [GUI_expression_format](#).
- <name>
(required) The name you define for the virtual signal.
Case is ignored unless installed in a Verilog region.
Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation.
If using VHDL extended identifier notation, <name> needs to be quoted with double quotes or with curly braces.

Examples

- Create a signal */chip/section1/clk_n* that is the inverse of */chip/section1/clk*.

```
virtual function { not /chip/section1/clk } clk_n
```
- Create a *std_logic_vector* equivalent of a Verilog register *rega* and installs it as */chip/rega_slv*.

```
virtual function -install /chip { (std_logic_vector) chip.vlog.rega  
} rega_slv
```
- Create a boolean signal */chip/addr_eq_fab* that is true when */chip/addr[11:0]* is equal to hex "fab", and false otherwise. It is acceptable to mix VHDL signal path notation with Verilog part-select notation.

```
virtual function { /chip/addr[11:0] == 0xfab } addr_eq_fab
```
- Create a signal that is high only during times when signal */chip/siga* of the gate-level version of the design does not match */chip/siga* of the rtl version of the design. Because there is no common design region for the inputs to the expression, *siga_diff* is installed in region *virtuals:/Functions*. The virtual function *siga_diff* can be added to the Wave window, and when expanded will show the two original signals that are being compared.

```
virtual function { gate:/chip/siga XOR rtl:/chip/siga } siga_diff
```
- Create a virtual signal consisting of the logical "AND" function of */top/signalA* with */top/signalB*, and delays it by 10 ns.

```
virtual function -delay {10 ns} {/top/signalA AND /top/signalB}  
myDelayAandB
```
- Create a one-bit signal *outbus_diff* which is non-zero during times when any bit of */chip/outbus* in the gate-level version doesn't match the corresponding bit in the rtl version.

This expression uses the "OR-reduction" operator, which takes the logical OR of all the bits of the vector argument.

```
virtual function { | (gate:/chip/outbus XOR rtl:/chip/outbus) }
outbus_diff
```

Commands fully compatible with virtual functions

add log and log	delete	describe
examine	find	restart
searchlog	show	
add list	add wave	checkpoint and restore
down and up	left and right	search

Commands not compatible with virtual functions

drivers	force	noforce
vcd add	when	
check contention add	check contention config	check contention off
check float add	check float config	check float off
check stable on	check stable off	power add
power report	power reset	toggle add
toggle reset	toggle report	

See also

virtual count	virtual define	virtual delete
virtual describe	virtual expand	virtual hide
virtual log	virtual nohide	virtual nolog
virtual region	virtual save	virtual show
virtual signal	virtual type	“Virtual Objects

virtual hide

The **virtual hide** command causes the specified real or virtual signals to not be displayed in the Objects window. This is used when you want to replace an expanded bus with a user-defined bus.

You make the signals reappear using the **virtual nohide** command.

Syntax

```
virtual hide { [-kind <kind>] | [-region <path>]} <pattern>
```

Arguments

- **-kind {implicits | explicits}**
(optional) Hides only a subset of virtuals.
 - implicits — virtual signals created internally by the tool.
 - explicits — virtual signals explicitly created by a user, such as with the virtual signal command.Unique abbreviations are accepted.
- **-region <path>**
(optional) Specifies a region of design space in which to look for the signal names.
- **<pattern>**
(required) Indicates which signal names or wildcard patterns should be used in finding the signals to hide, where wildcards are allowed and you can specify any number of names or patterns.

See also

[virtual nohide](#), [“Virtual Objects”](#)

virtual log

The **virtual log** command causes the simulation-mode dependent signals of the specified virtual signals to be logged by the kernel.

If wildcard patterns are used, it will also log any normal signals found, unless the **-only** option is used. You unlog the signals using the **virtual nolog** command.

Syntax

```
virtual log [-kind <kind>] | [-region <path>] [-recursive] [-only] [-in] [-out] [-inout] [-internal]
  [-ports] <pattern>
```

Arguments

- **-kind {implicits | explicits}**
(optional) Logs only a subset of virtuals.
 - implicits — virtual signals created internally by the tool.
 - explicits — virtual signals explicitly created by a user, such as with the virtual signal command.Unique abbreviations are accepted.
- **-region <path>**
(optional) Specifies a region of design space in which to look for signals to log.
- **-recursive**
(optional) Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region.
- **-only**
(optional) Specify that only virtual signals (as opposed to all signals) found by a <pattern> containing a wildcard should be logged.
- **-in**
Specifies that the kernel log data for ports of mode IN whose names match the specification. Optional.
- **-out**
(optional) Specifies that the kernel log data for ports of mode OUT whose names match the specification.
- **-inout**
(optional) Specifies that the kernel log data for ports of mode INOUT whose names match the specification.
- **-internal**
(optional) Specifies that the kernel log data for internal (non-port) objects whose names match the specification.

Commands

virtual log

- -ports
(optional) Specifies that the kernel log data for all ports. Optional.
- <pattern>
(required) Indicates which signal names or wildcard patterns should be used in finding the signals to log, where you can specify any number of names or wildcard patterns.

See also

[virtual nolog](#), [“Virtual Objects”](#)

virtual nohide

The **virtual nohide** command reverses the effect of a **virtual hide** command, causing the specified real or virtual signals to reappear the Objects window.

Syntax

```
virtual nohide { [-kind <kind>] | [-region <path>] } <pattern>
```

Arguments

- -kind { implicits | explicits }

(optional) Unhides only a subset of virtuals.

implicits — virtual signals created internally by the tool.

explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

Unique abbreviations are accepted.

- -region <path>

(optional) Specifies a region of design space in which to look for the signal names.

- <pattern>

(required) Indicates which signal names or wildcard patterns should be used in finding the signals to hide, where wildcards are allowed and you can specify any number of names or patterns.

See also

[virtual hide](#), “[Virtual Objects](#)”

virtual nolog

The **virtual nolog** command reverses the effect of a **virtual log** command. It causes the simulation-dependent signals of the specified virtual signals to be excluded ("unlogged") by the kernel.

If wildcard patterns are used, it will also unlog any normal signals found, unless the **-only** option is used.

Syntax

```
virtual nolog { [-kind <kind>] | [-region <path>]} [-recursive] [-only] [-in] [-out] [-inout]
               [-internal] [-ports] <pattern>
```

Arguments

- **-kind {implicits | explicits}**
(optional) Excludes only a subset of virtuals.
 - implicits — virtual signals created internally by the tool.
 - explicits — virtual signals explicitly created by a user, such as with the virtual signal command.Unique abbreviations are accepted.
- **-region <path>**
(optional) Specifies a region of design space in which to look for signals to unlog.
- **-recursive**
(optional) Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region.
- **-only**
(optional) Specify that only virtual signals (as opposed to all signals) found by a <pattern> containing a wildcard should be unlogged.
- **-in**
(optional) Specifies that the kernel exclude data for ports of mode IN whose names match the specification.
- **-out**
(optional) Specifies that the kernel exclude data for ports of mode OUT whose names match the specification.
- **-inout**
(optional) Specifies that the kernel exclude data for ports of mode INOUT whose names match the specification.

- **-internal**
(optional) Specifies that the kernel exclude data for internal (non-port) objects whose names match the specification.
- **-ports**
(optional) Specifies that the kernel exclude data for all ports.
- **<pattern>**
(required) Indicates which signal names or wildcard patterns should be used in finding the signals to unlog, where wildcards are allowed and you can specify any number of names or patterns.

See also

[virtual log](#), “[Virtual Objects](#)”

virtual region

The **virtual region** command creates a new user-defined design hierarchy region.

Syntax

```
virtual region <parentPath> <regionName>
```

Arguments

- <parentPath>
(required) The full path to the region that will become the parent of the new region.
- <regionName>
(required) The name you want for the new region.

See also

[virtual function](#), [virtual signal](#), “[Virtual Objects](#)”

Note



Virtual regions cannot be used in the [when](#) command.

virtual save

The **virtual save** command saves the definitions of virtuals to a file named `virtual.do` in the current directory.

Syntax

```
virtual save [-kind <kind>] [-append] [<filename>]
```

Arguments

- `-kind {implicits | explicits}`

(optional) Saves only a subset of virtuals.

`implicits` — virtual signals created internally by the tool.

`explicits` — virtual signals explicitly created by a user, such as with the virtual signal command.

Unique abbreviations are accepted.

- `-append`

(optional) Specifies to save **only** virtuals that are not already saved or weren't read in from a macro file. These unsaved virtuals are then appended to the specified or default file.

Optional.

- `<filename>`

(optional) The name of the file containing the definitions. If you don't specify `<filename>`, the default virtual filename (`virtuals.do`) will be used. You can specify a different default in the `pref.tcl` file.

See also

[virtual count](#), [“Virtual Objects”](#)

virtual show

The **virtual show** command lists the full path names of all explicitly defined virtuals.

Syntax

```
virtual show [-kind <kind>]
```

Arguments

- -kind {implicits | explicits}

(optional) Lists only a subset of virtuals.

implicits — virtual signals created internally by the tool.

explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

Unique abbreviations are accepted.

See also

[virtual define](#), [virtual describe](#), “[Virtual Objects](#)”

virtual signal

The **virtual signal** command creates a new signal, known only by the GUI (not the kernel), that consists of concatenations of signals and subelements as specified in `<expressionString>`.

It cannot handle bit selects and slices of Verilog registers. Please see [Concatenation of Signals or Subelements](#) for more details on syntax.

Note



The virtual function and virtual signal commands are interchangeable. The product will keep track of whether you've created a signal or a function with the commands and maintain them appropriately. We document both commands because the virtual save, virtual describe, and virtual define commands will reference your virtual objects using the correct command.

Syntax

```
virtual signal [-env <path>] [-install <path>] [-delay <time>] {<expressionString>} <name>
```

Arguments

- `-env <path>`
(optional) Specifies a hierarchical context for the signal names in `<expressionString>` so they don't all have to be full paths.
- `-install <path>`
(optional) Causes the newly-created signal to become a child of the specified region. If **-install** is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in `<expressionString>`. If the expression references more than one WLF file (dataset), the virtual signal will automatically be placed in region `virtuals:/Signals`.
- `-delay <time>`
(optional) Specifies a value by which the virtual function will be delayed. You can use negative values to look forward in time. If units are specified, the `<time>` option must be enclosed in curly braces. See the examples below for more details.
- `{<expressionString>}`
(required) A text string expression, enclosed in curly braces (`{ }`) using the [GUI_expression_format](#).
- `<name>`
(required) The name you define for the virtual signal.
Case is ignored unless installed in a Verilog region.
Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation.

If using VHDL extended identifier notation, **<name>** needs to be quoted with double quotes or with curly braces.

Examples

- Reconstruct a bus `sim:/chip/alu/a(4 downto 0)`, using VHDL notation, assuming that `a_ii` are all scalars of the same type.

```
virtual signal -env sim:/chip/alu { (concat_range (4 downto 0))(a_04
& a_03 & a_02 & a_01 & a_00) } a
```

- Reconstruct a bus `sim:chip.alu.a[4:0]`, using Verilog notation. Note that the concatenation notation starts with "&{" rather than "{".

```
virtual signal -env sim:chip.alu
{ (concat_range [4:0])&{a_04, a_03, a_02, a_01, a_00} } a
```

- Create a signal `sim:/testbench/stuff` which is a record type with three fields corresponding to the three specified signals. The example assumes `/chipa/mode` is of type integer, `/chipa/alu/a` is of type `std_logic_vector`, and `/chipa/decode/inst` is a user-defined enumeration.

```
virtual signal -install sim:/testbench
{ /chipa/alu/a(19 downto 13) & /chipa/decode/inst & /chipa/mode }
stuff
```

- Create a virtual signal that is the same as `/top/signalA` except it is delayed by 10 ps.

```
virtual signal -delay {10 ps} {/top/signalA} myDelayedSignalA
```

- Create a three-bit signal, `chip.address_mode`, as an alias to the specified bits.

```
virtual signal { chip.instruction[23:21] } address_mode
```

- Concatenate signals `a`, `b`, and `c` with the literal constant `'000'`.

```
virtual signal {a & b & c & 3'b000} myextendedbus
```

- Add three missing bits to the bus `num`, creates a virtual signal `fullbus`, and then adds that signal to the Wave window.

```
virtual signal {num & "000"} fullbus
add wave -unsigned fullbus
```

- Reconstruct a bus that was fragmented by synthesis and is missing the lower three bits. Note that you would have to type in the actual bit names (i.e. `num28`, `num27`, etc.) represented by the `...` in the syntax above.

```
virtual signal { num31 & num30 & num29 & ... & num4 & num3 & "000" }
fullbus
add wave -unsigned fullbus
```

- Create a two-bit signal (with an enumerated type) based on the results of the subexpressions. For example, if `aold` equals `anew`, then the first bit is true (1).

Alternatively, if *bold* does not equal *bnew*, the second bit is false (0). Each subexpression is evaluated independently.

```
virtual signal {(aold == anew) & (bold == bnew)} myequalityvector
```

- Create signal *newbus* that is a concatenation of bus1 (bit-reversed) and bus2[7:4] (bit-reversed). Assuming bus1 has indices running 7 downto 0, the result will be newbus[11:0] with the upper 8 bits being bus1[0:7] and the lower 4 bits being bus2[4:7]. See [Concatenation Directives](#) for further details.

```
virtual signal {(concat_reverse)(bus1 & bus2[7:4])} newbus
```

Commands fully compatible with virtual signals

add list	add log or log	add wave
delete	describe	examine
find	force and noforce	restart
searchlog	show	
checkpoint and restore	down and up	left and right
search		

Commands compatible with virtual signals using [virtual expand <signal>]

drivers	vcd add	
check contention add	check contention config	check contention off
check float add	check float config	check float off
check stable on	check stable off	
power add	power report	power reset
toggle add	toggle reset	toggle report

Commands not currently compatible with virtual signals

[when](#)

See also

virtual count	virtual define	virtual delete
virtual describe	virtual expand	virtual hide

virtual log

virtual region

virtual function

virtual nohide

virtual save

virtual type

virtual nolog

virtual show

“Virtual Objects

virtual type

The **virtual type** command creates a new enumerated type known only by the GUI, not the kernel. Virtual types are used to convert signal values to character strings. The command works with signed integer values up to 64 bits.

Virtual types cannot be used in the **when** command.

Note



If you are using SystemVerilog, you can also convert signal values to character strings using associative arrays in your code. See the SystemVerilog LRM for more information.

Syntax

```
virtual type -delete <name> | {<list_of_strings>} <name>
```

Arguments

- -delete <name>

Deletes a previously defined virtual type. <name> is the name you gave the virtual type when you originally defined it. Required if not defining a type.

- {<list_of_strings>}

A list of values and their associated character strings. Required if **-delete** is not used. Values can be expressed in decimal or based notation and can include "don't-cares" (see examples below). Three kinds of based notation are supported: Verilog, VHDL, and C-language styles. The values are interpreted without regard to the size of the bus to be mapped. Bus widths up to 64 bits are supported.

There is currently no restriction on the contents of each string, but if strings contain spaces they would need to be quoted, and if they contain characters treated specially by Tcl (square brackets, curly braces, backslashes...), they would need to be quoted with curly braces.

See the examples below for further syntax.

- <name>

The user-defined name of the virtual type. Required if **-delete** is not used. Case is not ignored. Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation. If using VHDL extended identifier notation, <name> needs to be quoted with double quotes or with curly braces.

Examples

- Using positional notation, associates each string with an enumeration index, starting at zero and increasing by one in the positive direction. When *myConvertedSignal* is displayed in the Wave, List, or Objects window, the string "state0" will appear when *mysignal* == 0, "state1" when *mysignal* == 1, "state2" when *mysignal* == 2, etc.

```
virtual type {state0 state1 state2 state3} mystateType
virtual function {(mystateType)mysignal} myConvertedSignal
add wave myConvertedSignal
```

- Use sparse mapping of bus values to alphanumeric strings for an 8-bit, one-hot encoding. It shows the variety of syntax that can be used for values. The value "default" has special meaning and corresponds to any value not explicitly specified.

```
virtual type {{0 NULL_STATE} {1 st1} {2 st2} {0x04 st3} {16'h08 st4} \
             {'h10 st5} {16#20 st6} {0b01000000 st7} {0x80 st8} \
             {default BAD_STATE}} myMappedType
virtual function {(myMappedType)mybus} myConvertedBus
add wave myConvertedBus
```

- Delete the virtual type "mystateType".

```
virtual type -delete mystateType
```

- Create a virtual type that includes "don't-cares" (the '-' character).

```
virtual type {{0x01-- add}{0x02-- sub}{default bad}} mydecodetype
```

- Create a virtual type using a mask for "don't-cares." The middle field is the mask, and the mask should have bits set to 1 for the bits that are don't care.

```
virtual type {{0x0100 0xff add}{0x0200 0xff sub}{default bad}}
mydecodetype
```

See also

[virtual function](#), [“Virtual Objects”](#)

vlib

The **vlib** command creates a design library. You must use **vlib** rather than operating system commands to create a library directory or index file.

If the specified library already exists as a valid ModelSim library, the **vlib** command will exit with a warning message without touching the library.

This command provides additional information with the **-help** switch.

Syntax

```
vlib [-archive [-compact <percent>]] [-format { 1 | 3 }] [-dos | -short | -unix | -long]
      [-lock | -unlock] [-locklib | -unlocklib] [-unnamed_designs <value>] <name>
```

Arguments

- -archive [-compact <percent>]

Causes design units that are compiled into the created library to be stored in archives rather than in subdirectories. Optional. Refer to “[Archives](#)” for more details.

You may optionally specify a decimal number between 0 and 1 that denotes the allowed percentage of wasted space before archives are compacted. By default archives are compacted when 50% (.5) of their space is wasted. See an example below.

- -format { 1 | 3 }

Prepares a library for conversion to be compatible with a previous release, by altering the `_info` file.

1 — allows you to convert a library to be compatible with the 6.2 series and earlier.

3 — allows you to convert a library to be compatible with the 6.3 series and newer.

The usage flow would be:

\1) Using a current release of the simulator, run:

```
vlib -format 1 current_lib
vcom -refresh -work current_lib
```

\| to prepare current_lib for conversion back to a 6.2 release

\|

\2) Using a 6.2 release of the simulator, run:

```
vcom -refresh -work current_lib
```

\| to refresh current_lib for use with the previous release

- -dos

Specifies that subdirectories in a library have names that are compatible with DOS. Not recommended if you use the **vmake** utility. Optional.

- -short

Interchangeable with the **-dos** argument. Optional.

- **-unix**
Specifies that subdirectories in a library may have long file names that are NOT compatible with DOS. Optional.
On by default for ModelSim SE.
- **-long**
Interchangeable with the **-unix** argument. Optional.
- **-lock | -unlock**
Locks an existing design unit so it cannot be recompiled or refreshed. The **-unlock** switch reverses this action. Optional. File permissions are not affected by these switches.
- **-locklib | -unlocklib**
Locks a complete library so that compilation cannot target the library and the library cannot be refreshed. The **-unlocklib** switch reverses this action. Optional. File permissions are not affected by these switches. Refer to the section “[Creating Locked Libraries for Multiple-User Simulation Environments](#)” in the User’s Manual for a scenario that uses these switches.
- **-unnamed_designs <value>**
Specifies how many unnamed, optimized versions of a design the **vopt** command will save within the library. Once **<value>** is reached, **vopt** deletes the oldest unnamed, optimized version. By default, the maximum number of “unnamed” designs (“_opt[number]”) is set to 3. Optional.
- **<name>**
Specifies the pathname or archive name of the library to be created. Required.

Examples

- Create the design library *design*. You can define a logical name for the library using the **vmap** command or by adding a line to the library section of the *modelsim.ini* file that is located in the same directory.

```
vlib design
```

- Create the design library *uut* and specifies that any design units compiled into the library are created as archives. Also specifies that each archive be compacted when 30% of its space is wasted.

```
vlib -archive -compact .3 uut
```

vlog

The **vlog** command compiles Verilog source code and SystemVerilog extensions into a specified working library (or to the **work** library by default).

The **vlog** command may be invoked from within ModelSim or from the operating system command prompt. It may also be invoked during simulation.

Compiled libraries are major-version dependent. When moving between major versions, you have to refresh compiled libraries using the **-refresh** argument to **vlog**. This is not true for minor versions (letter releases).

All arguments to the **vlog** command are case sensitive: **-WORK** and **-work** are not equivalent.

The IEEE P1800 Draft Standard for SystemVerilog requires that the default behavior of the **vlog** command is to treat each Verilog design file listed on the command line as a separate compilation unit. This behavior is a change in **vlog** from versions prior to 6.2, wherein all files in a single command line were concatenated into a single compilation unit. To treat multiple files listed within a single command line as a single compilation unit, use either the **vlog -mfcu** argument or the [MultiFileCompilationUnit](#) *modelsim.ini* file variable.

This command provides additional information with the **-help** switch.

Syntax

```
vlog [options] <filename> [<filename> ...]
```

[options]:

```
[-0in] [-0in_options "<options>"] [-32 | -64] [-93]
[+acc[=<spec>] [+<selection> [.]]
[-compat] [-compile_uselibs[=<directory_name>]]
  [-constimmedassert | -noconstimmedassert] [-convertallparams] [+cover[=<spec>]]
  [-coveropt <opt_level>] [-coverreportcancelled] [-covercells | -nocovercells] [-
  coverenhanced]
  [-coverexcludedefault] [-cuname]
[+define+<macro_name>[=<macro_text>]] [+delay_mode_distributed]
  [+delay_mode_path] [+delay_mode_unit] [+delay_mode_zero]
  [[-dpiforceheader]] [-dpiheader <filename>]
[-E <filename>] [-enumbaseinit] [-Epretty <filename>]
  [-error <msg_number>[,<msg_number>,...]] [-extendedtogglemode 1|2|3]
[-f <filename>] [+floatparameters[+<selection>[.]]] [-force_refresh <design_unit>]
  [-fsmimplicittrans] [-fsmmultitrans] [-fsmresettrans | -nofsmresettrans]
  [-fsmsingle | -nofsmsingle] [-fsmverbose[b | t | w]] [-fsmxassign | -nofsmxassign]
[-gen_xml <design_unit> <filename>]
[-hazards]
[+incdir+<directory>] [-incr | -noincr] [+initmem[=<spec>][+{0 | 1 | X | Z}]]
  [+initreg[=<spec>][+{0 | 1 | X | Z}]] [-isymfile]
```

`[-L <libname>] [-Lf <libname>] [+libcell] [+libext+<suffix>] [-libmap <pathname>]`
`[-libmap_verbos] [+librescan] [-line <number>] [-lint]`
`[-lowercasepragma] [-lowercasepslpragma]`

`[+maxdelays] [+mindelays] [-maxfecrows] [-maxudprows] [-mixedansiports]`
`[-mixedsvvh [b | s | v]] [-mfcu | -sfcu] [-modelsimini <ini_filepath>]`
`[-mti_trace_vlog_calls]`

`[-nodbgSYM] [-nocovershort] [-nocoverfec] [-nocoverudp]`
`[-noForceUnsignedToVhdlInteger] [+nolibcell] [-nologo] [-nops] [+nospecify]`
`[-note <msg_number>[,<msg_number>,...]] [-novopt] [+notimingchecks]`
`[-novtblfixup] [+nowarn<CODE>] [-nowarn <category_number>]`
`[-nodebug[=ports | =pli | =ports+pli]] [+nosparse[+<selection> [.]]]`
`[+num_opt_cell_conds+<value>]`

`[-oldsv] [-O0 | -O1 | -O4 | -O5]`

`[-pedanticerrors] [-permissive] [-permit_defunct_sv] [-printinfilenames] [-pslex]`
`[-pslfile <filename>] [+protect[=<filename>]]`

`[-quiet]`

`[-R [<simargs>]] [-refresh]`

`[-scdpiheader <filename>] [-source] [-s] [-sv] [-skipprotected] [-skipprotectedmodule]`
`[-suppress <msg_number>[,<msg_number>,...]] [-synthprefix <prefix>]`

`[-time] [-timescale <time_units>/<time_precision>] [-togglecountlimit <int>]`
`[-togglewidthlimit <int>] [+typdelays]`

`[-u]`

`[-v <library_file>] [-version] [-vlog01compat] [-vlog95compat] [-vmake]`
`[-vopt | -novopt]`

`[-warning <msg_number>[,<msg_number>,...]] [-work <library_name>]`
`[-writetoplevels <fileName>]`

`[-y <library_directory>]`

Arguments

- `-0in`
Invokes **0-In analyze** from within ModelSim. Optional. For the invocation to work, the environment variable HOME_0IN must be set to the proper pathname.
- `-0in_options "<options>"`
Passes the specified options to **0-In analyze**. Optional. The options must be enclosed in parentheses. Consult the 0-In documentation for details on the available options. For the invocation to work, the environment variable HOME_0IN must be set to the proper pathname.
- `-32 | -64`
Specifies whether vlog uses the 32- or 64-bit executable, where -32 is the default.

These options override the `MTI_VCO_MODE` environment variable, which applies only to executables used from the `<install_dir>/bin/` directory. Therefore, these options are ignored if you run `vlog` from an `<install_dir>/<platform>/` directory.

You can only specify these options on the command line, therefore they are not recognized as part of a file used with the `-f` switch.

- `-93`
Specifies that the VHDL interface to Verilog modules use VHDL 1076-1993 extended identifiers to preserve case in Verilog identifiers that contain uppercase letters. Optional.
- `+acc[=<spec>] [+<selection> [.]`
Enables PLI and debug command access to design objects that would otherwise become unavailable due to optimizations. Optional.

Note

Using this option may reduce optimizations.

`<spec>` is one or more of the following characters. If `<spec>` is omitted, the entire set of access specifiers is enabled.

- a —
Preserve PSL and SystemVerilog assertion and functional coverage browser data. Note that if a PSL or SystemVerilog construct is being driven by a port signal, **vopt** may replace that signal name with its higher-level driver. So in this case, if you prefer the local port name, the `+acc "p"` option should also be specified (i.e., `+acc=ap`).
- b —
Enable access to bits of vector nets. This is necessary for PLI applications that require handles to individual bits of vector nets. Also, some user interface commands require this access if you need to operate on net bits.
- c —
Enable access to library cells. By default any Verilog module containing a non-empty specify block may be optimized, and debug and PLI access may be limited. This option keeps module cell visibility.
- f —
Enable access to finite state machines.
- l —
Enable access to line number directives and process names.
- m —
Preserve the visibility of primitive gates.
- n —
Enable access to nets.

- p —
Enable access to ports. This disables the module inlining optimization, and is necessary only if you have PLI applications that require access to port handles.
- r —
Enable access to registers (including memories, integer, time, and real types).
- s —
Enable access to system tasks.
- t —
Enable access to tasks and functions.
- u —
Enable access to primitive instances.
- +<selection> — enables access for specific Verilog design objects and/or regions, optionally followed by ".", selection occurs recursively downward from the specified module or instance. Multiple selections are allowed, with each separated by a "+" (+acc=rn+top1+top2). If no selection is specified, then all modules are affected. Ensure that you do not put a space between any <spec> arguments and the +<selection> argument. You can use a path delimiter to select unique instances or objects (+acc=mrp+/top/ul. or +acc=r+/top/myreg). If you specify a module name (+acc=rn+Demux), pertinent objects in side the module are selected.

The tool determines whether you have supplied a module name or a object pathname by the existence of the "PathSeparator" character (set in the *modelsim.ini* file) in the path. By default, the separator is a '/'. An example object pathname specification is:

+acc=n+/u1/u2/n2

- -compat
Disables optimizations that result in different event ordering than Verilog-XL. Optional.
ModelSim Verilog generally duplicates Verilog-XL event ordering, but there are cases where it is inefficient to do so. Using this option does not help you find event order dependencies, but it allows you to ignore them. Keep in mind that this option does not account for all event order discrepancies, and that using this option may degrade performance. Refer to "[Event Ordering in Verilog Designs](#)" for additional information.
- -compile_uselib[=<directory_name>]
Locates source files specified in a **uselib** directive (Refer to "[Verilog-XL uselib Compiler Directive](#)"), compiles those files into automatically created libraries, and updates the *modelsim.ini* file with the logical mappings to the new libraries. Optional. If a *directory_name* is not specified, ModelSim uses the name specified in the MTI_USELIB_DIR environment variable. If that variable is not set, ModelSim creates the directory *mti_uselibs* in the current working directory.
- -constimmedassert
Displays immediate assertions with constant expressions in the GUI, in reports, and in the UCDB. Optional. By default, immediate assertions with constant expressions are displayed in the GUI, in reports, and in the UCDB. Use this switch only if the -noconstimmedassert

switch has been used previously, or if the ShowConstantImmediateAsserts variable in the vlog section of the *modelsim.ini* file is set to 0 (off).

- -noconstimedassert

Turns off the display of immediate assertions with constant expressions in the GUI, in reports, and in the UCDB. Optional. By default, immediate assertions with constant expressions are displayed. You may also set the ShowConstantImmediateAsserts variable in the vlog section of the *modelsim.ini* file to 0 (off).

- -convertallparams

Enables converting parameters not defined in ANSI style to VHDL generics of type std_logic_vector, bit_vector, std_logic, vl_logic, vl_logic_vector, and bit. Optional.

- +cover[=<spec>]

Enables various coverage statistics collection on all design units compiled in the current compiler run. Optional. Consider using the +cover argument to [vopt](#) instead, which you can use to specify precise design units and regions to be instrumented for coverage. The +cover argument with no "=<spec>" designation is equivalent to "+cover=bcesft".

<spec> — one or more of the following characters:

- b — Collect branch statistics.
- c — Collect condition statistics. Collects both FEC and UDP statistics, unless -nocoverfec or -nocoverudp is specified.
- e — Collect expression statistics, Collects both FEC and UDP statistics, unless -nocoverfec or -nocoverudp is specified.
- s — Collect statement statistics.
- t — Collect toggle statistics. Overridden if 'x' is specified elsewhere.
- x — Collect extended toggle statistics (Refer to “[Toggle Coverage](#)” for details). This takes precedence, if 't' is specified elsewhere.
- f — Collect Finite State Machine statistics.

See [-coveropt <opt_level>](#) argument to override the default level of optimization for coverage for a particular compilation run.

- -cover <spec>

Recommendation: Use "vopt +cover" rather than "vlog -cover", which you can use to specify precise design units and regions to be instrumented for coverage. See [vopt](#) for more information.

Specifies type(s) of coverage statistics to collect. Optional. <spec> is one or more of the following characters:

- b — Collect branch statistics.
- c — Collect condition statistics. Collects both FEC and UDP statistics, unless -nocoverfec or -nocoverudp is specified.

- e — Collect expression statistics, Collects both FEC and UDP statistics, unless `-nocoverfec` or `-nocoverudp` is specified.
- s — Collect statement statistics.
- t — Collect toggle statistics. Cannot be used if 'x' is specified.
- x — Collect extended toggle statistics (Refer to “[Toggle Coverage](#)” for details). Cannot be used if 't' is specified.
- f — Collect Finite State Machine statistics.

- `-covercells`

Enables code coverage of modules defined by `'celldefine` and `'endcelldefine` compiler directives, or compiled with the `-v` or `-y` arguments. Optional. Can be used to override the [CoverCells](#) compiler control variable in the `modelsim.ini` file.

- `-coverenhanced`

Enables non-critical functionality which might change the appearance or content of coverage metrics. This argument only has an effect in letter releases (6.6a, 6.6b, etc.) . In major releases (6.6, 6.7, etc.), all coverage enhancements present in previous letter release streams are enabled by default, and `-coverenhanced` is no longer necessary to enable these enhancements. Bug fixes important to the correctness of coverage numbers are always enabled by default, with no need for `-coverenhanced`. Since the exact nature of `-coverenhanced` varies from release to release, the details of the enhancements it enables are present in the product release notes rather than in the Command Reference. For these details, search the release notes using the string "coverenhanced".

- `-coverexcludedefault`

Excludes VHDL code coverage data collection from the OTHERS branch in both Case statements and Selected Signal Assignment statements. Optional.

- `-coveropt <opt_level>`

Overrides the default level of optimization for the current run only. Optional. `<opt_level>` designates the optimization level, as follows:

1 — Turns off all optimizations that affect coverage reports.

2 — Allows optimizations that provide large performance improvements by invoking sequential processes only when the data changes. This setting may result in major reductions in coverage counts.

3 — Allows all optimizations in 2, and allows optimizations that may change expressions or remove some statements. Also allows constant propagation and VHDL subprogram inlining.

4 — Allows all optimizations in 2 and 3, and allows optimizations that may remove major regions of code by changing assignments to built-ins or removing unused signals. It also changes Verilog gates to continuous assignments. Allows VHDL subprogram inlining. Allows VHDL flip-flop recognition.

The default optimization level is 3. You can edit the `CoverOpt` variable in the `modelsim.ini` file to change the default.

- `-coverreportcancelled`

(optional) Enables code coverage reporting of branch conditions that have been optimized away due to a static or null condition. The line of code is labeled EA in the hits column of the Source Window and EBCS in the hits column of a Coverage Report. You can also set this with the `CoverReportCancelled` `modelsim.ini` variable.

- `-cuname`

Used only in conjunction with `-mfcu`. Optional. The `-cuname` names the compilation unit being created by `vlog`. The named compilation unit can then be specified on the `vsim` command line, along with the `<top>` design unit. The purpose of doing so is to force elaboration of specified compilation unit package, thereby forcing elaboration of a necessary 'bind' statement within that compilation unit that would otherwise not be elaborated. An example of the necessary commands is:

```
vlog -cuname pkg_name -mfcu file1.sv file2.sv
vsim top pkg_name
```

You need to do this only in cases where you have a 'bind' statement in a module that might otherwise not be elaborated, because no module in the design depends on that compilation unit. In other words, if a module that depends on that compilation unit exists, you don't need to force the elaboration, for it occurs automatically. Also, if you are using qverilog to compile and simulate the design, this binding issue is handled properly automatically.

- `+define+<macro_name>[=<macro_text>]`

Allows you to define a macro from the command line that is equivalent to the following compiler directive:

```
`define <macro_name> <macro_text>
```

Optional. You can specify more than one macro with a single `+define`. For example:

```
vlog +define+one=r1+two=r2+three=r3 test.v
```

A command line macro overrides a macro of the same name defined with the ``define` compiler directive.

- `+delay_mode_distributed`

Disables path delays in favor of distributed delays. Optional. Refer to “[Delay Modes](#)” for details.

- `+delay_mode_path`

Sets distributed delays to zero in favor of using path delays. Optional.

- `+delay_mode_unit`

Sets path delays to zero and non-zero distributed delays to one time unit. Optional.

- `+delay_mode_zero`
Sets path delays and distributed delays to zero. Optional.
- `-dpiforceheader`
Forces the generation of a DPI header file even if it will be empty of function prototypes.
- `-dpiheader <filename>`
Generates a header file that may then be included in C source code for DPI import functions. Optional. Refer to “[DPI Use Flow](#)” for additional information.
- `-E <filename>`
Captures text processed by the Verilog parser after preprocessing has occurred and copies that text to an output file, `<filename>`. Optional. Generally, preprocessing consists of the following compiler directives: ``ifdef`, ``else`, ``elsif`, ``endif`, ``ifndef`, ``define`, ``undef`, ``include`.
The ``line` directive attempts to preserve line numbers and file names in the output file. White space is usually preserved, but sometimes it may be deleted or added to the output file.
- `-enumbaseinit`
Initializes enum variables in SystemVerilog using the default value of the base type instead of the leftmost value. Optional. Edit the `EnumBaseInit` variable in the `modelsim.ini` file to set a permanent default.
- `-Epretty <filename>`
Captures text processed by the Verilog parser after preprocessing has occurred, performs some formatting for better readability, and copies that text to an output file, `<filename>`. Optional.
- `-error <msg_number>[,<msg_number>,...]`
Changes the severity level of the specified message(s) to "error." Optional. Edit the `error` variable in the `modelsim.ini` file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- `-extendedtogglemode 1|2|3`
Changes the level of support for extended toggles. Optional. The levels of support are:
 - 1 — 0L->1H & 1H->0L & any one 'Z' transition (to/from 'Z')
 - 2 — 0L->1H & 1H->0L & one transition to 'Z' & one transition from 'Z'
 - 3 — 0L->1H & 1H->0L & all 'Z' transitionsEdit the `ExtendedToggleMode` variable in the `modelsim.ini` file to set a permanent default.
- `-f <filename>`
Specifies a file with more command line arguments. Optional. Allows complex arguments to be reused without retyping. Allows gzipped input files. Nesting of `-f` options is allowed. Refer to the section “[Argument Files](#)” for more information.

- +floatparameters[+<selection>[.]]

Instructs the tool to not lock down parameter values during optimization, which enables successful use of the vsim -g/G options.

+<selection> — localizes the effect of this option to specific parameters in the design hierarchy. +<selection> can be a hierarchical path to a parameter or a design unit instance. It can also be the name of a design unit declaration.

. — If a period (.) is present after an instance or design unit name, all parameters under that scope are recursively selected.

- -force_refresh <design_unit>

Forces the refresh of all specified design units. Optional. By default, the work library is updated; use **-work <library_name>**, in conjunction with **-force_refresh**, to update a different library (for example, vlog -work <your_lib_name> -force_refresh).

When the compiler refreshes a design unit, it checks each dependency to ensure its source has not been changed and recompiled. Sometimes the dependency checking algorithm changes from release to release. This can lead to false errors during the integrity checks performed by the **-refresh** argument. An example of such a message follows:

```
** Error: (vsim-13) Recompile /u/test/dware/dware_61e_beta.dwpackages  
because /home/users/questasim/linux/./synopsys.attributes has changed.
```

The **-force_refresh** argument forces the refresh of the design unit, overriding any dependency checking errors encountered by the **-refresh** argument.

A more conservative approach to working around **-refresh** dependency checks is to recompile the source code, if it is available.

- -fsmimplicitrans

Enables recognition of implied same state transitions. Optional.

- -fsmmultitrans

Enables detection and reporting of multi-state transitions when used with the +cover=f argument for vlog or vopt. Optional. Another term for this is FSM sequence coverage.

- -fsmresettrans

Enables recognition of implicit asynchronous reset transitions. Optional. This includes asynchronous reset transitions in coverage results.

- -fsmsingle | -nofsmsingle

(optional) Enables the recognition of VHDL FSMs where the current state variable of type std_logic, bit, boolean, or single-bit std_logic_vector/bit_vector and Verilog single-bit FSMs.

- -fsmverbose[b | t | w]

Provides information about FSMs detected, including state reachability analysis. Optional.

This switch only provides this data when you use the -novopt switch on the same command line.

- b — displays only basic information.
- t — displays a transition table in addition to the basic information.
- w — displays any warning messages in addition to the basic information.

When you do not specify an argument, this switch reports all information similar to:

```
# ** Note: (vlog-1947)   FSM RECOGNITION INFO
#   Fsm detected in : ../fpu/rtl/vhdl/serial_mul.vhd
#   Current State Variable : s_state :
#   ../fpu/rtl/vhdl/serial_mul.vhd(76)
#   Clock : clk_i
#   Reset States are: { waiting , busy }
#   State Set is : { busy , waiting }
#   Transition table is
#   -----
#   busy      =>  waiting Line : (114 => 114)
#   busy      =>  busy   Line : (111 => 111)
#   waiting   =>  waiting Line : (120 => 120) (114 => 114)
#   waiting   =>  busy   Line : (111 => 111)
#   -----
```

When you do not specify this switch, you will receive a message similar to:

```
# ** Note: (vlog-143) Detected '1' FSM/s in design unit
# 'serial_mul.rtl'.
```

- **-fsmxassign**

Enables recognition of finite state machines (FSMs) containing X assignment. Optional. This option is used to detect FSMs if current state variable or next state variable has been assigned "X" value in a "case" statement. FSMs containing X-assign are otherwise not detectable.

- **-gen_xml <design_unit> <filename>**

Produces an XML-tagged file containing the interface definition of the specified module. Optional. This option requires a two-step process where you must 1) compile <filename> into a library with **vlog** (without **-gen_xml**) then 2) execute **vlog** with the **-gen_xml** switch, for example:

```
vlib work
vlog counter.v
vlog -gen_xml counter counter.xml
```

- **-hazards**

Detects event order hazards involving simultaneous reading and writing of the same register in concurrently executing processes. Optional. You must also specify this argument when you simulate the design with **vsim**. Refer to [“Hazard Detection”](#) for more details.

Note



Enabling **-hazards** implicitly enables the **-compat** argument. As a result, using this argument may affect your simulation results.

- `+incdir+<directory>`

Specifies directories to search for files included with ``include` compiler directives. Optional. By default, the current directory is searched first and then the directories specified by the `+incdir` options in the order they appear on the command line. You may specify multiple `+incdir` options as well as multiple directories separated by "+" in a single `+incdir` option.

- `-incr`

Performs an incremental compile. Optional. Default. Compiles only code that has changed. For example, if you change only one module in a file containing several modules, only the changed module will be recompiled. Note however that if the compile options change, all modules are recompiled regardless if you use `-incr` or not.

- `+initmem[=<spec>][+{0 | 1 | X | Z}]`

Enables the initialization of memories. Optional.

`<spec>` — (optional) identifies the types to be initialized.

If you do not specify this option, vlog initializes fixed-size arrays of all these types, where fixed-size arrays may have any number of packed or unpacked dimensions.

`<spec>` can be one or more of the following:

r — register/logic, integer, or time types (four-state integral types).

b — bit, int, shortint, longint, or byte types (two-state integral types).

e — enum types.

You must also add the enum's base type to the initialization specification. If you choose static initialization for an enum type variable with value 0, 1, X, or Z, the simulator assigns that value to the variable, whether it is a valid value or not. If you choose random initialization for an enum type variable, the simulator generates a random number and uses the `(random_number % num_valid_enum_values)`th entry of the enum literals to initialize it.

u — sequential UDPs.

`+{0 | 1 | X | Z}` — (optional) specifies the value to use in initialization for all bits of a memory. For two-state datatypes, X and Z will map to 0.

If you do not specify this option you are preparing the design unit for randomization with `vsim +initmem +<seed>`.

This argument initializes static variables in any scope (package, \$unit, module, interface, generate, program, task, function). However, it does not affect:

- automatic variables
- dynamic variables
- members of dynamic variables
- artificially generated variables, such as `#randstate#`

Because you can specify `+initmem` on the `vlog` and `vopt` command line, the priority of the specifications are as follows:

- 1) `vopt ... +initmem+1+top.foo`
- 2) `vlog ... +initmem+0`
- 3) `vopt ... +initmem+Z`

This argument will not override any variable declaration assignment, such as:

```
reg r = 1'b0
```

- `+initreg[=<spec>][+{0 | 1 | X | Z}]`

Enables you to initialize registers. Optional.

`<spec>` — identifies the types to be initialized.

If you do not specify this option, `vlog` initializes variables of all these types.

`<spec>` can be one or more of the following:

`r` — register/logic, integer, or time types (four-state integral types).

Notifier registers are not initialized by the `+initreg` option. To detect that a register is a notifier, timing checks must be present, which means you cannot compile with the `+nospecify` or `+notimingchecks` arguments. However, if you want to remove timing checks but still detect notifier registers, use `vsim +notimingchecks` or `vsim +nospecify`. You can also do this is by using ``ifdef` to remove timing checks.

`b` — bit, int, shortint, longint, or byte types (two-state integral types).

`e` — enum types.

You must also add the enum's base type to the initialization specification. If you choose static initialization for an enum type variable with value 0, 1, X, or Z, the simulator assigns that value to the variable, whether it is a valid value or not. If you choose random initialization for an enum type variable, the simulator generates a random number and uses the $(\text{random_number} \% \text{num_valid_enum_values})\text{th}$ entry of the enum literals to initialize it.

`u` — sequential UDPs.

If a sequential UDP contains an "initial" statement, that initial value overrides all `+initreg`-related functionality. For other sequential UDPs, the `+initreg` option takes effect as described for regular variables. In case a sequential UDP does not contain an "initial" statement, and it wasn't compiled with `+initreg` in effect, the UDPs initial value will be taken from its instantiating parent scope (provided that scope has `+initreg` options in effect).

`+{0 | 1 | X | Z}` — (optional) specifies the value to use in initialization. For two-state datatypes, X and Z will map to 0.

If you do not specify this option you are preparing the design unit for randomization with `vsim +initreg +<seed>`

This argument initializes static variables in any scope (package, \$unit, module, interface, generate, program, task, function). However, it does not affect:

- automatic variables
- dynamic variables
- members of dynamic variables
- artificially generated variables, such as `#randstate#`

Because you can specify `+initreg` on the `vlog` and `vopt` command line, the priority of the specifications are as follows:

- 1) `vopt ... +initreg+1+top.foo`
- 2) `vlog ... +initreg+0`
- 3) `vopt ... +initreg+Z`

This argument will not override any variable declaration assignment, such as:

```
reg r = 1'b0
```

- `-isymfile`

Generates a complete list of all imported tasks and functions (TFs). Used with DPI to determine all imported TFs that are expected by ModelSim.

- `-L <libname>`

Searches the specified resource library for precompiled modules. The library search options you specify here must also be specified when you run the `vsim` command. Optional. See also the [LibrarySearchPath](#) variable and [Specifying the Resource Libraries in the User's Manual](#).

Note



The `-L` argument is required when you are importing a UPF package in a Power Aware RTL (PA-RTL) flow. You must specify `mtiUPF` as the value of `<libname>`.

- `-Lf <libname>`

Same as `-L`, but the specified library is searched before any `'uselib` directives. (Refer to “[Library Usage](#)” and “[Verilog-XL Compatible Compiler Arguments](#)” for more information.) Optional.

- `+libcell`

Treats all modules found and compiled by source library search as though they contained a `'celldefine` compiler directive, thus marking them as cells (refer to the `-v` and `-y` arguments of `vlog`, which enable source library search). Using the `+libcell` argument matches historical behavior of Verilog-XL with respect to source library search. Optional.

Note



By default, wildcard logging and code coverage exclude cells. For more information, refer to the `-nocovercells` and `-covercells` arguments of `vlog` and to the description of wildcard logging performed by the `log` command.

- `+libext+<suffix>`

Works in conjunction with the `-y` option. Specifies file extensions for the files in a source library directory. Optional. By default, the compiler searches for files without extensions. If you specify the `+libext` argument, then the compiler will search for a file with the suffix appended to an unresolved name. You may specify only one `+libext` option, but it may contain multiple suffixes separated by the plus character (+). The extensions are tried in the order you specify them with the `+libext` argument.

- `-libmap <pathname>`

Specifies a Verilog 2001 library map file. Optional. You can omit this argument by placing the library map file as the first option on the `vlog` invocation (e.g., `vlog top.map top.v top_cfg.v`).

- `-libmap_verbose`

Displays library map pattern matching information during compilation. Optional. Use to troubleshoot problems with matching filename patterns in a library file.

- `+librescan`

Scans libraries in command-line order for all unresolved modules. Optional.

- `-line <number>`

Starts the compiler on the specified line in the Verilog source file. Optional. By default, the compiler starts at the beginning of the file.

- `-lint`

(optional) Issues warnings on the following lint-style static checks:

- when Module ports are NULL.
- when assigning to an input port.
- when referencing undeclared variables/nets in an instantiation.

This switch generates additional array bounds-checking code, which can slow down simulation, to check for the following:

- index warnings for dynamic arrays
- when an index for a Verilog unpacked variable array reference is out of bounds.

The warnings are reported as `WARNING[8]`. You can also enable this option using the [Show_Lint](#) variable in the `modelsim.ini` file.

- `-lowercasepragma`

Accepts only lower case pragmas in Verilog source files. Optional. You can also enable this feature by setting the [AcceptLowerCasePragmaOnly](#) variable in the `modelsim.ini` file.

- `-lowercasepslpragma`

Forces the Verilog compiler to accept only lower case (embedded) PSL pragmas.

- **+maxdelays**
Selects maximum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.
- **+mindelays**
Selects minimum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.
- **-maxfecrows**
Sets the maximum number of rows allowed in an FEC truth table for a code coverage condition or expression. The default maximum is 192 rows, which allows for 96 terms in the expression. Increasing the number of rows includes more expressions for coverage, but also increases the compile time, sometimes dramatically. You can also configure this option using the [CoverMaxFECRows](#) variable in the *modelsim.ini* file.
- **-maxudprows**
Sets the maximum number of rows allowed in an UDP truth table for a code coverage condition or expression. The default maximum is 192 rows. Increasing the number of rows includes more expressions for coverage, but also increases the compile time, sometimes dramatically. You can also configure this option using the [CoverMaxUDPRows](#) variable in the *modelsim.ini* file.
- **-mixedansiports**
Permits partial port redeclarations.
- **-mixedsvvh [b | s | v]**
Facilitates using SystemVerilog packages at the SystemVerilog-VHDL boundary of a mixed-language design. When you compile a SystemVerilog package with **-mixedsvvh**, the package can be included in a VHDL design as if it were defined in VHDL itself. Optional.
 - b** — treats all scalars/vectors in the package as VHDL bit/bit_vector
 - s** — treats all scalars/vectors in the package as VHDL std_logic/std_logic_vector
 - v** — treats all scalars/vectors in the package as VHDL vl_logic/vl_logic_vector
- **-mfcu**
Instructs the compiler to treat all files within a compilation command line as a single compilation unit. Optional. The default behavior is to treat each file listed in a command as a separate compilation unit, per the SystemVerilog standard. Prior versions concatenated the contents of the multiple files into a single compilation unit by default. You can also enable this option using the [MultiFileCompilationUnit](#) variable in the *modelsim.ini* file.
- **-modelsimini <ini_filepath>**
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or

relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

- **-mti_trace_vlog_calls**

Enables viewing of SystemVerilog class contents in the Wave window. Optional.

- **-nocovercells**

Disables code coverage of modules defined by `'celldefine` and `'endcelldefine` compiler directives, or compiled with the `-v` or `-y` arguments. Optional. Can be used to override the [CoverCells](#) compiler control variable in the `modelsim.ini` file.

- **-nodebug[=`ports` | `=pli` | `=ports+pli`]**

Hides, within the GUI and other parts of the tool, the internal data of all compiled design units. Optional.

-nodebug — The switch, specified in this form, does not hide ports, due to the fact that the port information may be required for instantiation in a parent scope.

The design units' source code, internal structure, registers, nets, etc. will not display in the GUI. In addition, none of the hidden objects may be accessed through the Dataflow or Schematic window or with commands. This also means that you cannot set breakpoints or single step within this code. It is advised that you not compile with this switch until you are done debugging.

Note that this is not a speed switch like the “nodebug” option on many other products.

Use the [vopt](#) command to increase simulation speed.

-nodebug=`ports` — additionally hides the ports for the lower levels of your design; it should be used only to compile the lower levels of the design. If you hide the ports of the top level you will not be able to simulate the design.

Do not use the switch in this form when the parent is part of a [vopt -bbox](#) flow or for mixed language designs, especially for Verilog modules to be instantiated inside VHDL.

-nodebug=`pli` — additionally prevents the use of `pli` functions to interrogate individual modules for information.

You should be aware that this form will leave a "nodebug" module untraversable by PLI.

-nodebug=`ports+pli` — you can combine the behavior of `=ports` and `=pli` in this manner.

This functionality encrypts entire files. The ``protect` compiler directive allows you to encrypt regions within a file.

- **-nodbsym**

Disables the generation of the symbols debugging database in the compiled library.

The symbols debugging database is the `.dbs` file in the compiled library that provides information to the GUI allowing you to view detailed information about design objects at the source level. Two major GUI features that use this database include source window annotation and textual dataflow.

You should only specify this switch if you know that anyone using the library will not require this information for design analysis purposes.

- **-nocoverfec**

Prevents focused expression coverage (FEC) from being enabled for coverage collection. By default, both UDP and FEC coverage statistics are enabled for collection. You can customize the default behavior with the [CoverFEC](#) variable in the *modelsim.ini* file. Optional.

- **-nocoverudp**

Prevents UDP coverage from being enabled for coverage collection. By default, both UDP and FEC coverage statistics are enabled for collection. You can customize the default behavior with the [CoverUDP](#) variable in the *modelsim.ini* file. Optional.

- **-nocovershort**

Disables short circuiting of expressions when coverage is enabled. Short circuiting is enabled by default. You can customize the default behavior with the [CoverShortCircuit](#) variable in the *modelsim.ini* file. Optional.

- **-noForceUnsignedToVhdlInteger**

Prevents untyped Verilog parameters in mixed-language designs that are initialized with unsigned values between $2^{*31}-1$ and 2^{*32} from being converted to a VHDL generic. By default, untyped Verilog parameters that are initialized with unsigned values between $2^{*31}-1$ and 2^{*32} are converted to VHDL INTEGER generics. Because VHDL INTEGER parameters are signed numbers, the Verilog values $2^{*31}-1$ to 2^{*32} are converted to negative VHDL values in the range from -2^{*31} to -1 (the 2's complement value).

- **-nofsmresettrans**

Disables recognition of implicit asynchronous reset transitions. Optional. This has the effect of excluding asynchronous reset transitions from any coverage results.

- **-nofsmxassign**

Disables recognition of FSMs containing x assignment. Optional.

- **-noincr**

Disables incremental compile previously turned on with **-incr**. Optional.

- **+nolibcell**

Disables treating all modules found and compiled by source library search as though they contained a 'celldefine compiler directive. That is, this argument restores the default library search behavior if you have changed it using the [+libcell](#) argument. Optional.

- **-nologo**

Disables the startup banner. Optional.

- -nopsl

Causes the compiler to ignore embedded PSL assertions. By default, vlog parses any PSL assertion statements it finds in the specified files. Optional. Refer to “[Compiling and Simulating Assertions](#)” for more information.

- +nosparse[+<selection> [.]]

Identifies which memories are considered “not sparse,” which causes ModelSim to override the rules for allocating storage for memory elements only when necessary. Optional.

If you use +nosparse on a given memory, the tool will simulate the memory normally. Refer to [Sparse Memory Modeling](#) for more information.

+<selection> — enables access for specific Verilog design units, scopes, or design objects (vars, mem). Multiple selections are allowed, with each separated by a "+" (+nosparse=+top1+top2). If no selection is specified, then all modules are affected. You can use a path delimiter to select unique instances or objects (+nosparse=+/top/ul.). If you specify a module name (+nosparse=+Demux), pertinent objects inside the module are selected.

. — indicates the selection occurs recursively downward from the specified module or instance.

- +nospecify

Disables specify path delays and timing checks. Optional.

- -note <msg_number>[,<msg_number>,...]

Changes the severity level of the specified message(s) to "note." Optional. Edit the `note` variable in the `modelsim.ini` file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

- +notimingchecks

Removes all timing check entries from the design as it is parsed. Optional.

To disable checks on individual instances, use the [tcheck_set](#) command.

- -novopt

Forces **vlog** to produce code if the [VoptFlow](#) variable is set to 1 (optimizations turned on) in the `modelsim.ini`. (VoptFlow = 1 is the default behavior.) Optional. Use this argument together with the **vsim -novopt** command to run the simulator without any optimizations. For example, you may want to use this argument when you are coding an RTL block with a small testcase.

- -novtblfixup

Causes virtual method calls in SystemVerilog class constructors to behave as they would in normal class methods, which prevents the type of a `this` reference from changing during construction.

This overrides default behavior, where the type of a `this` reference is treated as if it is a handle to the type of the active `new()` method while a constructor is executing (which

implies that virtual method calls resolve will not execute methods of an uninitialized class type).

- +nowarn<CODE>

Disables warning messages in the category specified by <CODE>. Optional. Warnings that can be disabled include the <CODE> name in square brackets in the warning message. For example,

```
** Warning: test.v(15): [RDGN] - Redundant digits in numeric
literal.
```

This warning message can be disabled by specifying +nowarnRDGN.

- -nowarn <category_number>

Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. Multiple **-nowarn** switches are allowed. Warnings may be disabled for all compiles via the Main window **Compile > Compile Options** menu command or the *modelsim.ini* file (refer to [modelsim.ini Variables](#)).

The warning message categories are described in [Table 2-11](#):

Table 2-11. Warning Message Categories for vlog -nowarn

Category number	Description
11	PSL warnings
12	non-LRM compliance in order to match Cadence behavior
13	constructs that code coverage can't handle
15	System Verilog assertions using local variable

- +num_opt_cell_conds+<value>

Restricts gate level optimization capacity for accepting cells with I/O path and timing check conditions. Optional.

value — integer between 32 and 1023, inclusive. where the default value is 1023.

- -oldsv

Allows white space between the grave accent mark (‘) and the opening brace of aggregate expressions. Optional.

- -O0 | -O1 | -O4 | -O5

Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

Please refer to the section "[Optimizing Designs with vopt](#)" in the User's Manual for detailed information on using vopt to perform optimization.

- Enable PE-level optimization with **-O1**. Optional.
- Enable standard SE optimizations with **-O4**. Default.
- Enable maximum optimization with **-O5**. Optional. **-O5** attempts to optimize loops and prevents variable assignments in situations where a variable is assigned but is not actually used. Using the **+acc** argument to **vlog** will cancel this latter optimization.
- **-pedanticerrors**

Enforces strict compliance of the IEEE Std 1800-2005 in the following cases:

- Using "new" for queues is not legal. When strict compliance is not enforced, use of "new" creates a queue of the specified size where all elements are initialized to the default value of the queue element type.
- Using underscore character (`_`) in sized, based literals is not legal. When you specify this argument, an error will occur for literals such as `2'b_01`.
- Omitting the grave accent mark (```) preceding the left brace when writing structure literals is not legal. When you specify this argument, an error will occur for literals written without that mark.
- Using class extern method prototypes with lifetime (automatic/static) designations. When you specify this switch, this scenario produces an LRM-compliance error, otherwise you will receive a warning.
- Using "cover bool@clk" as a PSL statement.

This argument also produces a report of mismatched 'else directives. Optional.

- **-permissive**
Allows messages in the LRM group of error messages to be downgraded to a warning. Optional.
- **-permit_defunct_sv**
Allows using a selected set of constructs no longer supported by the SystemVerilog standard. Currently, the set supports only the use of the keyword "char." This argument allows use of the keyword "char" to be interpreted as the SystemVerilog "byte" type. Optional.
- **-printinfilenames**
Prints the path names of all source files opened (including "include" files) during the compile. Specifies whether each file is a Verilog or SystemVerilog file.
- **+protect[=<filename>]**
Enables ``pragma protect`, ``protect`, and ``endprotect` directives for encrypting selected regions of your source code. Optional. Produces an encrypted output file with a `.vp` extension in the default work directory. To create an encrypted output file to the current directory, add `=<filename>` to this argument. If you specify a filename, all source files on the command line are concatenated together into a single output file.

Any ``include` files will also be inserted into the output file when you add `=<filename>`. If you do not use `=<filename>`, all ``include` files will be encrypted into the work directory as individual files, not merged together into one file.

- `-pslxt`
Enables PSL LTL and OBE operators. Optional. These operators are disabled by default.
- `-pslfile <filename>`
Identifies an external PSL assertion file to compile along with the Verilog source files. Refer to “[Compiling and Simulating Assertions](#)” for more information.
- `-quiet`
Disables 'Loading' messages. Optional.
- `-R [<simargs>]`
Instructs the compiler to invoke [vsim](#) after compiling the design. The compiler automatically determines which top-level modules are to be simulated. The command line arguments following `-R` are passed to the simulator, not the compiler. Place the `-R` option at the end of the command line or terminate the simulator command line arguments with a single `"-"` character to differentiate them from compiler command line arguments.

The `-R` option is not a Verilog-XL option, but it is used by ModelSim to combine the compile and simulate phases together as you may be used to doing with Verilog-XL. It is not recommended that you regularly use this option because you will incur the unnecessary overhead of compiling your design for each simulation run. Mainly, it is provided to ease the transition to ModelSim.
- `-refresh`
Regenerates a library image. Optional. By default, the work library is updated. To update a different library, use `-work <library_name>` with `-refresh` (for example, `vlog -work <your_lib_name> -refresh`). If a dependency checking error occurs which prevents the refresh, use the `vlog -force_refresh` argument. See [vlog](#) examples for more information. You may use a specific design name with `-refresh` to regenerate a library image for that design, but you may not use a file name.
- `-scdpiheader <filename>`
Specifies the name of SystemC DPI function prototype header file automatically generated from the current compilation. Optional. The default filename is `sc_dpiheader.h` when no such switch is provided. Refer to section [SystemC Procedural Interface to SystemVerilog](#) for more detailed description.
- `-sfcu`
Instructs the compiler to treat all files within a compilation command line as a separate compilation units. This is the default behavior and is the inverse of the behavior of `-mfcu`.

This switch will override the [MultiFileCompilationUnit](#) variable if it is set to "1" in the `modelsim.ini` file.

- **-source**
Displays the associated line of source code before each error message that is generated during compilation. Optional; by default, only the error message is displayed.
- **-s**
Instructs the compiler not to load the **standard** package. Optional. This argument should only be used when you are compiling the **sv_std** package.
- **-sv**
Enables SystemVerilog features and keywords. Optional. By default ModelSim follows the rules of IEEE Std 1364-2001 and ignores SystemVerilog keywords. If a source file has a ".sv" extension, ModelSim will automatically parse SystemVerilog keywords.
- **-skipprotected**
Ignores any 'protect/'endprotect region contained in a module. Optional.
- **-skipprotectedmodule**
Prevents adding any module containing a 'protect/'endprotect region to the library. Optional.
- **-suppress <msg_number>[,<msg_number>,...]**
Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress Fatal or Internal messages. Edit the **suppress** variable in the *modelsim.ini* file to set a permanent default. Refer to "[Changing message Severity Level](#)" for more information.
- **-synthprefix <prefix>**
(optional) Enables recognition of synthesis pragmas with a user specified prefix. If this argument is not specified, pragmas are treated as comments and the previously excluded statements included in the synthesized design. All regular synthesis pragmas are honored.

 <prefix> — Specifies a user defined string.
- **-time**
Reports the "wall clock time" **vlog** takes to compile the design. Optional. Note that if many processes are running on the same system, wall clock time may differ greatly from the actual "cpu time" spent on **vlog**.
- **-timescale <time_units>/<time_precision>**
Specifies the default timescale for modules not having an explicit timescale directive in effect during compilation. Optional. The format of the **-timescale** argument is the same as that of the ``timescale` directive. The format for <time_units> and <time_precision> is <n><units>. The value of <n> must be 1, 10, or 100. The value of <units> must be fs, ps, ns, us, ms, or s. In addition, the <time_units> must be greater than or equal to the <time_precision>.

- `-togglecountlimit <int>`

Limits the toggle coverage count, `<int>`, for a toggle node. Optional. After the limit is reached, further activity on the node is ignored for toggle coverage. All possible transition edges must reach this count for the limit to take effect. For example, if you are collecting toggle data on 0->1 and 1->0 transitions, both transition counts must reach the limit. If you are collecting "full" data on 6 edge transitions, all 6 must reach the limit. Overrides the global value set by the `ToggleCountLimit` *modelsim.ini* variable.
- `-togglewidthlimit <int>`

Sets the maximum width of signals, `<int>`, that are automatically added to toggle coverage with the `-cover t` argument. Optional. Can be set on design unit basis. Overrides the global value of the `ToggleWidthLimit` *modelsim.ini* variable.
- `+typdelays`

Selects typical delays from the "min:typ:max" expressions. Default. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.
- `-u`

Converts regular Verilog identifiers to uppercase. Allows case insensitivity for module names. Optional.
- `-v <library_file>`

Specifies a source library file containing module and UDP definitions. Optional. Refer to “[Verilog-XL Compatible Compiler Arguments](#)” for more information.

After all explicit filenames on the `vlog` command line have been processed, the compiler uses the `-v` option to find and compile any modules that were referenced but not yet defined. Modules and UDPs within the file are compiled only if they match previously unresolved references. Multiple `-v` options are allowed. See additional discussion in the examples.
- `-version`

Returns the version of the compiler as used by the licensing tools. Optional.
- `-vlog01compat`

Ensures compatibility with rules of IEEE Std 1364-2001. Default.
- `-vlog95compat`

Disables Verilog 2001 keywords, which ensures that code that was valid according to the 1364-1995 spec can still be compiled. By default ModelSim follows the rules of IEEE Std 1364-2001. Some requirements in 1364-2001 conflict with requirements in 1364-1995. Optional. Edit the `vlog95compat` variable in the *modelsim.ini* file to set a permanent default.
- `-vopt`

Notifies `vlog` that the `vopt` command will be run. As a result, `vlog` does not produce code. This saves an unnecessary code generation step. Optional.

This argument is needed only if you have set the `VoptFlow` variable to 0 in the *modelsim.ini* file. If it is set to 1 (default operation) `vlog` skips the code generation step automatically.

- -vmake

Generates a complete record of all command line data and files accessed during the compile of a design. This data is then used by the [vmake](#) command to generate a comprehensive makefile for recompiling the design library. By default, vcom stores compile data needed for the -refresh switch and ignores compile data not needed for -refresh. The -vmake switch forces inclusion of all file dependencies and command line data accessed during a compile, whether they contribute data to the initial compile or not. Executing this switch can increase compile time in addition to increasing the accuracy of the compile. See the vmake command for more information.

- -warning <msg_number>[,<msg_number>,...]

Changes the severity level of the specified message(s) to "warning." Optional. Edit the [warning](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

- -work <library_name>

Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional; by default, the compiled design units are added to the **work** library. The specified pathname overrides the pathname specified for work in the project file.

- -writetoplevels <fileName>

(optional) Records the names of all top level module names in a specified file. Also records any compilation unit name specified with **-cuname**. May only be specified when compiling the top level modules.

<fileName> — Required. Specifies the name of the file where module names are to be recorded.

- -y <library_directory>

Specifies a source library directory containing definitions for modules, packages, interfaces, and user-defined primitives (UDPs). Usually, this is a directory of source files that you want to scan if the compiled versions do not already exist in a library. Optional. Refer to “[Verilog-XL Compatible Compiler Arguments](#)” for more information.

After all explicit filenames on the **vlog** command line have been processed, the compiler uses the **-y** option to find and compile any modules that were referenced but not yet defined. Files within this directory are compiled only if the file names match the names of previously unresolved references. Multiple **-y** options are allowed. You will need to specify a file suffix by using **-y** in conjunction with the **+libext+<suffix>** option if your filenames differ from your module names. See additional discussion in the examples.

Note



Any **-y** arguments that follow a **-refresh** argument on a **vlog** command line are ignored. Any **-y** arguments that come before the **-refresh** argument on a **vlog** command line are processed.

- `<filename>`
Specifies the name of the Verilog source code file to compile. One filename is required. Multiple filenames can be entered separated by spaces. Wildcards can be used.

Examples

- Compile the Verilog source code contained in the file *example.vlg*.

```
vlog example.vlg
```

- Hide the internal data of *example.v*. Models compiled with **-nodebug** cannot use any of the ModelSim debugging features; any subsequent user will not be able to see into the model.

```
vlog -nodebug example.v
```

- The first line compiles and hides the internal data, plus the ports, of the lower-level design units, *level3.v* and *level2.v*. The second line compiles the top-level unit, *top.v*, without hiding the ports. It is important to compile the top level without **=ports** because top-level ports must be visible for simulation.

```
vlog -nodebug=ports level3.v level2.v  
vlog -nodebug top.v
```

The first command hides the internal data, and ports of the design units, *level3.v* and *level2.v*. In addition it prevents the use of PLI functions to interrogate the compiled modules for information (either **=ports+pli** or **=pli+ports** works fine for this command). The second line compiles the top-level unit without hiding the ports but restricts the use of PLI functions as well.

- Note that the **=pli** switch may be used at any level of the design but **=ports** should only be used on lower levels since you can't simulate without visible top-level ports.

```
vlog -nodebug=ports+pli level3.v level2.v  
vlog -nodebug=pli top.v
```

- After compiling *top.v*, **vlog** will scan the file *und1* for modules or primitives referenced but undefined in *top.v*. Only referenced definitions will be compiled.

```
vlog top.v -v und1
```

- After compiling *top.v*, **vlog** will scan the *vlog_lib* library for files with modules with the same name as primitives referenced, but undefined in *top.v*. The use of **+libext+.v+.u** implies filenames with a *.v* or *.u* suffix (any combination of suffixes may be used). Only referenced definitions will be compiled.

```
vlog top.v +libext+.v+.u -y vlog_lib
```

The **-work** option specifies **mylib** as the library to regenerate. **-refresh** rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of ModelSim.

- If your library contains VHDL design units, be sure to regenerate the library with the `vcom` command using the **-refresh** option as well. Refer to “[Regenerating Your Design Libraries](#)” for more information.

```
vlog -work mylib -refresh
```

- The **-incr** option determines whether or not the module source or compile options have changed as *module1.v* is parsed. If no change is found, the code generation phase is skipped. Differences in compile options are determined by comparing the compiler options stored in the `_info` file with the compiler options given. They must match exactly.

```
vlog module1.v -u -O0 -incr
```

- The **-timescale** option specifies the default timescale for *module1.v*, which did not have an explicit timescale directive in effect during compilation. Quotes are necessary because the argument contains white spaces.

```
vlog module1.v -timescale "1 ns / 1 ps"
```

- The **-fsmmultitrans** option enables detection and reporting of multi-state transitions when used with the `+cover f` argument.

```
vlog +cover=f -fsmmultitrans
```

vmake

The **vmake** utility allows you to use a UNIX or Windows MAKE program to maintain individual libraries. You run **vmake** on a compiled design library. This utility operates on multiple source files per design unit; it supports Verilog include files as well as Verilog and VHDL PSL vunit files.

Note



If a design is spread across multiple libraries, then each library must have its own makefile and you must build each one separately.

By default, the output of **vmake** is sent to stdout—however, you can send the output to a makefile by using the shell redirect operator (>) along with the name of the file. You can then run the makefile with a version of MAKE (not supplied with ModelSim) to reconstruct the library. *This command must be invoked from either the UNIX or the Windows/DOS prompt.*

A MAKE program is included with Microsoft Visual C/C++, as well as many other program development environments.

After running the **vmake** utility, MAKE recompiles only the design units (and their dependencies) that have changed. You run **vmake** only once; then you can simply run MAKE to rebuild your design. If you add new design units or delete old ones, you should re-run **vmake** to generate a new makefile.

The **vmake** utility ignores library objects compiled with **-nodebug**.

Also, the **vmake** utility is not supported for use with SystemC.

This command provides additional information with the **-help** switch.

Syntax

```
vmake [-cygdrive] [-du <design_unit_name> ...] [-f <filename>] [-fullsrcpath] [-ignore]
      [<library_name>] [-modelsimini <ini_filepath>]
```

Arguments

- **-cygdrive**
Generates a makefile that uses a path specified with UNIX pathname conventions. Use this argument if you are using cygwin v3.81 or later (which no longer supports Windows conventions for drive and pathname). Optional.
- **-du <design_unit_name>**
Specifies that a vmake file will be generated only for the specified design unit. You can specify this argument any number of times for a single vmake command. Optional.
- **-f <filename>**
Specifies a file to read command line arguments from. Optional.
Refer to the section "[Argument Files](#)" for more information

- **-fullsrcpath**
Produces complete source file paths within generated makefiles. By default, source file paths are relative to the directory in which compilations originally occurred. Use this argument to copy and evaluate generated makefiles within directories that are different from where compilations originally occurred. Optional.
- **-ignore**
Omits a make rule for the named primary design unit and its secondary design units. Optional.
- **<library_name>**
Specifies the library name; if none is specified, then work is assumed. Optional.
- **-modelsimini <ini_filepath>**
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified by the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems, the path separator should be a forward slash (/). Optional.

Examples

- To produce a makefile for the work library:

```
vmake >mylib.mak
```

- To run **vmake** on libraries other than **work**:

```
vmake mylib >mylib.mak
```

- To rebuild **mylib**, specify its makefile when you run MAKE:

```
make -f mylib.mak
```


vmap

The **vmap** command defines a mapping between a logical library name and a directory by modifying the *modelsim.ini* file.

With no arguments, **vmap** reads the appropriate *modelsim.ini* file(s) and prints to the transcript the current logical library to physical directory mappings.

This command provides additional information with the **-help** switch.

Syntax

```
vmap [-c | -del <logical_name> ... | <logical_name> [<path>] ]
```

Arguments

- **-c**
(optional) Copies the default *modelsim.ini* file from the ModelSim installation directory to the current directory.

This argument is intended only for making a copy of the default *modelsim.ini* file to the current directory. Do not use it while making your library mappings or the mappings may end up in the incorrect copy of the *modelsim.ini*.
- **-del <logical_name> ...**
(optional) Deletes the mapping specified by <logical_name> from the current project file. You can specify multiple logical name arguments to the **-del** switch to delete multiple library mappings.
- **<logical_name> [<path>]**
(optional) Maps a logical library name to the specified physical library.

If you do not specify <path> the command returns the current mapping for <logical_name>.

Examples

- Map two logical libraries to the physical library “work”, then delete the two logical libraries:

```
vlib work
vmap library1 work
vmap library2 work
```

- Display information about the logical library “library1”:

```
vmap library1
```

- Delete the logical library mappings:

```
vmap -del library1 library2
```

vopt

The **vopt** command performs global optimizations on designs after they have been compiled with **vcom** or **vlog**. For detailed usage information on optimization, refer to the chapter titled “[Optimizing Designs with vopt](#)” in the User’s Manual.

Note



The default optimization behavior of **vopt** may differ from what is documented in this section if you are using any *modelsim.ini* file other than the one you installed with ModelSim. See “[Optimizing Designs with vopt](#)” and “[VoptFlow](#)” in the User’s Manual for information on possible optimization settings.

The **vopt** command produces an optimized version of your design in the working directory. You provide a name for this optimized version using the **-o** argument. You can then invoke **vsim** directly on that new design unit name.

The **vopt** command makes use of the [PathSeparator](#) variable from the *modelsim.ini* file, which allows you to specify which character that ModelSim recognizes as a path separator. By default, **vopt** expects you to use the forward-slash (/), but you could change this to a period (.) for ease of use with Verilog designs. For example:

```
vopt top -o opttop +acc=rn+.top.middle.bottom -G.top.myparam=4
```

The default operation of **vopt -o <name>** is incremental compilation – that is, ModelSim reuses elements of the design that have not changed. This improves performance when using **vopt** on a design that has been minimally modified.

In the course of optimizing a design, **vopt** will remove objects that are deemed unnecessary for simulation. For example, line numbers are removed, processes are merged, nets and registers may be removed, etc. If you need visibility into your design for debugging purposes, use the **+acc** argument to conditionally enable visibility for parts of your design. Note, however, that using **+acc** may reduce simulation speed.

All arguments to the **vopt** command are case sensitive: **-WORK** and **-work** are not equivalent.

Note



Many of the arguments below may also be specified at compile time (e.g., **-O5**). These “in-common” arguments are processed by **vopt** automatically (i.e., if you specify them at compile time, **vopt** will use them automatically during optimization). The first instance of an in-common argument takes priority. For example, if you specify **-O5** to **vlog** or **vcom**, and then specify **-O1** to **vopt**, **-O5** takes precedence.

There is one exception to this rule: **vopt** will “OR” any **+cover** arguments to **vlog** or **vcom** (**vlog +cover=bc**) with any **+cover** arguments to **vopt** (**vopt -cover=est**)

This command provides additional information with the **-help** switch.

Syntax

vopt [options] <design_unit> -o <name>

[options]:

[-0in] [-0in_options "<options>"] [-32 | -64]
 [+acc[=<spec>][+<selection>[. | +<entity>[([architecture](#))] [.]]]
 [-bbox]
 [+check<enum>] [-compat] [-constimmedassert | -noconstimmedassert]
 [-cpppath <filename>] [+cover[=<spec>] [+<selection> [.]]] [-cover <spec>]
 [-coverenhanced] [-coverexcludedefault] [-coveropt <opt_level>]
 [-coverreportcancelled] [-nocovershort] [-covercells]
 [-enablescsdout]
 [-debugCellOpt] [-deferSubpgmCheck] [+delay_mode_distributed]
 [+delay_mode_path]
 [+delay_mode_unit] [+delay_mode_zero] [-dpiforceheader] [-dpiheader <filename>]
 [-error <msg_number>[,<msg_number>,...]] [-extendedtogglemode 1|2|3]
 [-f <filename>] [-fatal <msg_number>[,<msg_number>,...]]
 [+floatgenerics[+<selection>[.]]] [+floatparameters[+<selection>[.]]]
 [-fsmimplicittrans] [-fsmmultitrans] [-fsmresettrans] [-fsmsingle | -nofsmsingle]
 [-fsmverbose [b | t | w]]
 [-g <Name>=<Value> ...] [-G <Name>=<Value> ...]
 [-hazards]
 [-incr | -noincr] [+initmem[=<spec>][+{0 | 1 | X | Z}][+<selection>[.]]]
 [+initreg[=<spec>][+{0 | 1 | X | Z}][+<selection>[.]]]
 [-ka] [-keep_delta]
 [-L <libname>] [-Lf <libname>] [-lowercasepslpragma]
 [+maxdelays] [+mindelays] [-mti_trace_vlog_calls]
 [-mixedansiports] [-modelsimini <ini_filepath>]
 [-nocover] [-nocovercells] [-nocoverfec] [-nocoverudp]
 [-nodebug[=ports | =pli | =ports+pli]] [-nofsmresettrans]
 [-nofsmxassign] [+nolibcell] [-nologo] [+nosparse[+<selection> [.]]] [+nospecify]
 [-note <msg_number>[,<msg_number>,...]] [+notimingchecks] [+nowarn<CODE>]
 [-nowarn <category_number>] [+num_opt_cell_conds+<value>]
 [-O0 | -O1 | -O4 | -O5] [-ocf <filename>]
 [-permissive] [-pslxt] [-pslfile_vh <filename>] [-pslfile_vl <filename>]
 [-quiet]
 [-sclib <library>]
 [-sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=<sdf_filename>]]

vopt

```

[-sdfmaxerrors <n>] [-source] [-suppress <msg_number>[,<msg_number>,...]]
[+staticchecks[=<args>]] [-staticchecksfile <filename>] [-staticchecksmdvhdl]
[-tab <tabfile>] [-time] [-timescale <time_units>/<time_precision>] [+typdelays]
[-version]
[-warning <msg_number>[,<msg_number>,...]] [-work <library_name>]

```

Using vopt in a Power Aware Verification Flow

The following optional arguments are specific to [Power Aware Verification](#).

```

vopt <design_unit> -o <name> -pa_all | -pa_out
[-pa_allowzerotimeevent] [-pa_bbox <dut_name>] [-pa_behavlogfile <filename>]
[-pa_ce {o | os | osw}] [-pa_cfg <filename>] [-pa_dbgstatic msk | rsn]
[-pa_checks={rop | rpo | rcs | rsa | r | iep | idp | irc | it | isa | i | t | cp | p | a | npu | ugc | upc | s |
sdl | sil | sml | snl | srl | svl | ul | uil | uml}]
[-pa_checkseq="<t1> [<unit>] [...<tn>]" ] [-pa_crptlogfile <filename>]
[-pa_dbgstatic msk | rsn] [-pa_dumpupf <filename>]
[[-pa_excludefile <filename> [-pa_enhexclude]]
[-pa_genrpt={ {nv | v} [ {us | ud | u} ] [b]}] [-pa_gls] [-pa_hiersep <alphanum_character>]
[-pa_intcrptval0] [-pa_lib <library_pathname>] [-pa_libertylibs=<database_pathname>]
[-pa_lsthreshold <real>] [-pa_modeltype=2] [-pa_nopcfctrlcheck]
[-pa_opt={0 | 1 | 2 | 6 | 7}] [[-pa_prefix <hier_path> [-pa_replacetop <string>]]
[-pa_reportfile <filename>] [-pa_rtinfo] [-pa_tclfile <filename>] [-pa_upf <filename>]
[-pa_upfscopeconns] [-pa_upfsyntaxchecks]

```

Arguments

- **-0in**
Invokes **0-In ccl** from within ModelSim. Optional. For the invocation to work, the environment variable HOME_0IN must be set to the proper pathname. For more information on **0-In ccl**, refer to 0-In documentation.
- **-0in_options "<options>"**
Passes the specified options to **0-In ccl**. Optional. The options must be enclosed in parentheses. Consult the 0-In documentation for details on the available options. For the invocation to work, the environment variable HOME_0IN must be set to the proper pathname. For more information on **0-In ccl**, refer to 0-In documentation.
- **-32 | -64**
Specifies whether vopt uses the 32- or 64-bit executable, where -32 is the default.
These options override the MTI_VCO_MODE environment variable, which applies only to executables used from the *<install_dir>/bin/* directory. Therefore, these options are ignored if you run vopt from an *<install_dir>/<platform>/* directory.
You can only specify these options on the command line, therefore they are not recognized as part of a file used with the -f switch.

- +acc[=<spec>][+<selection>[.] | +<entity>[(<architecture>)] [.]

Enable PLI and debug command access to objects indicated by <spec> when optimizing a design. Optional.

Note

Using this option may reduce simulation speed.

=<spec> — optionally, one or more of the following characters. If <spec> is omitted, the entire set of access specifiers is enabled.

- a — Preserve PSL and SystemVerilog assertion and functional coverage data. Enables pass count logging in the Transcript window and assertion viewing in the Wave window. Also enables the complete functionality of vsim -assertdebug. If you do not specify +acc=a, the tool only transcripts assertion failure messages and reports only failure counts in the assertion browser. Note that if a PSL or SystemVerilog construct is being driven by a port signal, **vopt** may replace that signal name with its higher-level driver. So in this case, if you prefer the local port name, the +acc "p" option should also be specified (i.e., +acc=ap).
- b — Enable access to bits of vector nets. This is necessary for PLI applications that require handles to individual bits of vector nets. Also, some user interface commands require this access if you need to operate on net bits.
- c — Enable access to library cells. By default any Verilog module containing a non-empty specify block may be optimized, and debug and PLI access may be limited. This option keeps module cell visibility.
- f — Enable access to finite state machines.
- l — Enable access to line number directives and process names.
- m — Preserve the visibility of module, program, and interface instances.
- n — Enable access to nets.
- p — Enable access to ports. This disables the module inlining optimization, and is necessary only if you have PLI applications that require access to port handles.
- r — Enable access to registers (including memories, integer, time, and real types).
- s — Enable access to system tasks.
- t — Enable access to tasks and functions.
- u — Enable access to primitive instances.
- v — Enable access to variables, constants, and aliases in processes (VHDL design units only) that would otherwise be merged due to optimizations. Disables an optimization that automatically converts variables to constants.

+<selection> — enables access for specific Verilog design objects and/or regions, optionally followed by ".", selection occurs recursively downward from the specified

module or instance. Multiple selections are allowed, with each separated by a "+" (+acc=rn+top1+top2). If no selection is specified, then all modules are affected. Ensure that you do not put a space between any <spec> arguments and the +<selection> argument. You can use a path delimiter to select unique instances or objects (+acc=mrp+/top/ul. or +acc=r+/top/myreg). If you specify a module name (+acc=rn+Demux), pertinent objects in side the module are selected.

+<entity>[(architecture)] — enables access for all instances of the specified VHDL entity, optionally followed by "." to indicate all children of the module.

- -bbox

Instructs vopt to optimize a portion of the design, allowing you to reuse the optimized portion and speed up future simulation and optimization runs. Refer to the section "[Optimizing Portions of your Design](#)" for further information.

- +check<enum>

Inhibits various modes of cell-level optimization checking, where <enum> represents one of the modes listed below. Optional.

ALL — Enables all +check modes described below.

AWA — Prevents an Always block from driving another Always block or a sequential UDP.

CLUP — Disables optimization of cells containing connectivity loops.

DELAY — When used in conjunction with vopt +delay_mode_path (see below), disables optimization of Verilog modules with distributed delays and no path delays in favor of module inlining.

DNET — Prevents both the port and the delayed port (created for negative setup/hold) to be used in the functional section of the cell.

INTRI — Disallows inputs of type tri1 or tri0 in cell optimizations.

IPDOP — Disallows procedural code where an input directly drives an output.

NWOT — Disallows a notifier register to be written outside of a timing check.

OPRD — Disallows an output port to be read internally by the cell.

SUDP — Disallows a sequential UDP to drive another sequential UDP.

- -compat

Disables optimizations that result in different event ordering than Verilog-XL. Optional.

ModelSim Verilog generally duplicates Verilog-XL event ordering, but there are cases where it is inefficient to do so. Using this option does not help you find event order dependencies, but it allows you to ignore them. Keep in mind that this option does not account for all event order discrepancies, and that using this option may degrade performance. Refer to "[Event Ordering in Verilog Designs](#)" for additional information.

- **-constimmedassert**
Displays immediate assertions with constant expressions in the GUI, in reports, and in the UCDB. Optional. By default, immediate assertions with constant expressions are displayed in the GUI, in reports, and in the UCDB. Use this switch only if the `-noconstimmedassert` switch has been used previously, or if the `ShowConstantImmediateAsserts` variable in the `vopt` section of the `modelsim.ini` file is set to 0 (off).
- **-cpppath <filename>**
Specifies the location of a g++ executable other than the default g++ compiler installed with ModelSim. Optional. Overrides the **CppPath** variable in the `modelsim.ini` file.
- **-noconstimmedassert**
Turns off the display of immediate assertions with constant expressions in the GUI, in reports, and in the UCDB. Optional. By default, immediate assertions with constant expressions are displayed. You may also set the `ShowConstantImmediateAsserts` variable in the `vopt` section of the `modelsim.ini` file to 0 (off).
- **+cover[=<spec>] [+<selection> [.]]**
Enables various coverage statistics collection on specified areas of the design. Optional.
Use this argument to enable coverage for specific design units or instances of the design. The `vopt +cover` argument accepts the same `+<selection>` syntax that `+acc` accepts, and can be used to specify design units, instances, and recursive control with a trailing '.' character.
`<spec>` — one or more of the following characters:
 - b — Collect branch statistics.
 - c — Collect condition statistics. Collects both FEC and UDP statistics, unless `-nocoverfec` or `-nocoverudp` is specified.
 - e — Collect expression statistics, Collects both FEC and UDP statistics, unless `-nocoverfec` or `-nocoverudp` is specified.
 - s — Collect statement statistics.
 - t — Collect toggle statistics. Overridden if 'x' is specified elsewhere.
 - x — Collect extended toggle statistics (Refer to “[Toggle Coverage](#)” for details). This takes precedence, if 't' is specified elsewhere.
 - f — Collect Finite State Machine statistics.`<selection> [.]` — path to the design unit or instance, with an optional "." to specify recursive coverage down to the leaf level.
See the `-coveropt <opt_level>` argument to override the default level of optimization for coverage for a particular compilation run.
Example:
`vopt +cover=bcest+/top/dut1. +cover=f+/top/dut1/fsm1 +cover=x+pla`

This command enables branch, condition, expression, statement and toggle coverage for instance */top/dut1* and all its children, down to the leaf level. It also turns on FSM coverage for only the */top/dut1/fsm1* instance. Finally, it enables extended toggle coverage for all instances of design unit *pla*.

- -cover <spec>

Recommendation: Use the +cover argument, which you can use to specify precise design units and regions to be instrumented for coverage.

Instructs ModelSim to recompile the design for specified types of coverage statistics collection. Optional.

<spec> — one or more of the following characters:

b — Collect branch statistics.

c — Collect condition statistics.

e — Collect expression statistics.

s — Collect statement statistics.

t — Collect toggle statistics. Cannot be used if 'x' is specified.

x — Collect extended toggle statistics (Refer to “[Toggle Coverage](#)” for details). Cannot be used if 't' is specified.

f — Collect Finite State Machine statistics.

<i> — Override the default level of optimization for current run only, where “i” is an integer between 1 and 4. To change default level for all subsequent runs, change value of [CoverOpt](#) variable in *modelsim.ini* file. See “[CoverOpt](#)” for a description of optimization levels.

When you specify -cover to vopt, it will logically OR the -cover arguments to vlog/vcom with the -cover arguments to vopt.

You can force coverage to be off and the time of running vopt by specifying [-nocover](#).

- -nocovershort

Disables short circuiting of expressions when coverage is enabled. Short circuiting is enabled by default.

- -covercells

Enables code coverage of modules defined by 'celldefine and 'endcelldefine compiler directives, or compiled with the -v or -y arguments. Optional. Can be used to override the [CoverCells](#) compiler control variable in the *modelsim.ini* file.

- -coverenhanced

Enables non-critical functionality which might change the appearance or content of coverage metrics. This argument only has an effect in letter releases (6.6a, 6.6b, etc.) . In major releases (6.6, 6.7, etc.), all coverage enhancements present in previous letter release streams are enabled by default, and -coverenhanced is no longer necessary to enable these enhancements. Bug fixes important to the correctness of coverage numbers are always

enabled by default, with no need for `-coverenhanced`. Since the exact nature of `-coverenhanced` varies from release to release, the details of the enhancements it enables are present in the product release notes rather than in the Command Reference. For these details, search the release notes using the string "coverenhanced".

- `-coverexcludedefault`

Excludes VHDL code coverage data collection from the OTHERS branch in both Case statements and Selected Signal Assignment statements. Optional.

- `-coveropt <opt_level>`

Overrides the default level of optimization for the current run only. Optional. `<opt_level>` designates the optimization level, as follows:

1 — Turns off all optimizations that affect coverage reports.

2 — Allows optimizations that provide large performance improvements by invoking sequential processes only when the data changes. This setting may result in major reductions in coverage counts.

3 — Allows all optimizations in 2, and allows optimizations that may change expressions or remove some statements. Also allows constant propagation and VHDL subprogram inlining.

4 — Allows all optimizations in 2 and 3, and allows optimizations that may remove major regions of code by changing assignments to built-ins or removing unused signals. It also changes Verilog gates to continuous assignments. Allows VHDL subprogram inlining. Allows VHDL flip-flop recognition.

The default optimization level is 3. You can edit the `CoverOpt` variable in the `modelsim.ini` file to change the default.

- `-coverreportcancelled`

(optional) Enables code coverage reporting of branch conditions that have been optimized away due to a static or null condition. The line of code is labeled EA in the hits column of the Source Window and EBCS in the hits column of a Coverage Report. You can also set this with the `CoverReportCancelled` `modelsim.ini` variable.

- `-debugCellOpt`

Produces Transcript window output that identifies why certain cells within the design were not optimized. Optional.

- `-deferSubpgmCheck`

Forces the compiler to report array indexing and length errors as warnings (instead of as errors) when encountered within subprograms. Subprograms with indexing and length errors that are invoked during simulation cause the simulator to report errors, which can potentially slow down simulation because of additional checking.

- `+delay_mode_distributed`

Disables path delays in favor of distributed delays. Optional. Refer to “[Delay Modes](#)” for details.

vopt

- **+delay_mode_path**
Sets distributed delays to zero in favor of using path delays. Optional.
- **+delay_mode_unit**
Sets path delays to zero and non-zero distributed delays to one time unit. Optional.
- **+delay_mode_zero**
Sets path delays and distributed delays to zero. Optional.
- **-dpiforceheader**
Forces the generation of a DPI header file even if it will be empty of function prototypes.
- **-dpiheader <filename>**
Generates a header file that may then be included in C source code for DPI import functions. Optional. Refer to “[DPI Use Flow](#)” for additional information.
- **-enablesstdout**
Enables the reporting of messages from the SystemC tasks cout, printf and fprintf to stdout during the execution of vopt. This behavior is suppressed by default. However, information printed to stderr will always be displayed.
- **-error <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "error." Optional. Edit the [error](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-extendedtogglemode 1|2|3**
Changes the level of support for extended toggles. Optional. The levels of support are:
 - 1 — 0L->1H & 1H->0L & any one 'Z' transition (to/from 'Z')
 - 2 — 0L->1H & 1H->0L & one transition to 'Z' & one transition from 'Z'
 - 3 — 0L->1H & 1H->0L & all 'Z' transitionsEdit the [ExtendedToggleMode](#) variable in the *modelsim.ini* file to set a permanent default.
- **-f <filename>**
Specifies a file with more command line arguments. Optional. Allows complex arguments to be reused without retyping. Allows gzipped input files. Nesting of **-f** options is allowed. Refer to the section “[Argument Files](#)” for more information.
- **-fatal <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "fatal." Optional. Edit the [fatal](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

- +floatgenerics[+<selection>[.]]

Instructs the tool to not lock down generics values during optimization, which enables successful use of the vsim -g/G options.

+<selection> — localizes the effect of this option to specific generics in the design hierarchy. +<selection> can be a hierarchical path to a generic or a design unit instance. It can also be the name of a design unit declaration.

. — If a period (.) is present after an instance or design unit name, all generics under that scope are recursively selected.

Refer to the section "[Optimizing Parameters and Generics](#)" for more information.

This command is fully equivalent to +floatparameters, therefore you can use them interchangeably.

- +floatparameters[+<selection>[.]]

Instructs the tool to not lock down parameter values during optimization, which enables successful use of the vsim -g/G options.

+<selection> — localizes the effect of this option to specific parameters in the design hierarchy. +<selection> can be a hierarchical path to a parameter or a design unit instance. It can also be the name of a design unit declaration.

. — If a period (.) is present after an instance or design unit name, all parameters under that scope are recursively selected.

Refer to the section "[Optimizing Parameters and Generics](#)" for more information.

This command is fully equivalent to +floatgenerics, therefore you can use them interchangeably.

- -fsmimplicitrans

Enables recognition of implied same state transitions. Optional.

- -fsmmultitrans

Enables detection and reporting of multi-state transitions when used with the +cover=f argument for [vcom/vlog](#) or vopt. Optional. Another term for this is FSM sequence coverage.

- -fsmresettrans

Enables recognition of implicit asynchronous reset transitions. Optional. This includes asynchronous reset transitions in coverage results.

- -fsmsingle | -nofmsingle

(optional) Enables the recognition of VHDL FSMs where the current state variable of type std_logic, bit, boolean, or single-bit std_logic_vector/bit_vector and Verilog single-bit FSMs.

- -fsmverbose [b | t | w]

Provides information about FSMs detected, including state reachability analysis. Optional.

b — displays only basic information.

t — displays a transition table in addition to the basic information.

w — displays any warning messages in addition to the basic information.

When you do not specify an argument, this switch reports all information to vopt message 1947.

```
# ** Note: (vopt-1947)   FSM RECOGNITION INFO
#       Fsm detected in : ../fpu/rtl/vhdl/serial_mul.vhd
#       Current State Variable : s_state :
#       ../fpu/rtl/vhdl/serial_mul.vhd(76)
#       Clock : clk_i
#       Reset States are: { waiting , busy }
#       State Set is : { busy , waiting }
#       Transition table is
#       -----
#       busy      =>  waiting Line : (114 => 114)
#       busy      =>  busy     Line : (111 => 111)
#       waiting   =>  waiting Line : (120 => 120) (114 => 114)
#       waiting   =>  busy     Line : (111 => 111)
#       -----
```

When you do not specify this switch, vopt reports only that an FSM was detected in vopt message 143.

```
# ** Note: (vopt-143) Detected '1' FSM/s in design unit
# 'serial_mul.rtl'.
```

- -g <Name>=<Value> ...

Assigns a value to all specified VHDL generics and Verilog parameters that have not received explicit values in generic maps, instantiations, or via defparams (such as top-level generics/parameters and generics/parameters that would otherwise receive their default values). Optional.

Note there is a space between -g and <Name>=<Value>. For more information on this switch, refer to the longer description under vsim -g<Name>=<Value> ...

Refer to the section "[Optimizing Parameters and Generics](#)" for more information.

Limitation: In general, generics/parameters of composite type (arrays and records) cannot be set from the command line. However, you can set string arrays, std_logic vectors, and bit vectors if they can be set using a quoted string. For example,

```
-g strgen="This is a string"
-g slv="01001110"
```

The quotation marks must make it into vsim as part of the string because the type of the value must be determinable outside of any context. Therefore, when entering this command from a shell, put single quotes (') around the string. For example:

```
-g strgen='This is a string'
```

If working within the ModelSim GUI, you would enter the command as follows:

```
{-g strgen="This is a string"}
```

You can also enclose the value escaped quotes (\"), for example:

```
-g strgen=\"This is a string\"
```

- `-G <Name>=<Value> ...`

Same as `-g` (see above) except that it will also override generics/parameters that received explicit values in generic maps, instantiations, or from defparams. Optional. This argument is the only way for you to alter the generic/parameter, such as its length, (other than its value) after the design has been loaded.

Note there is a space between `-G` and `<Name>=<Value>`.

If `<value>` is a string, you must enclose it in escaped quotes (`\`), for example:

```
-G filename=\"a.in\"
```

Refer to the section "[Optimizing Parameters and Generics](#)" for more information.

- `-hazards`

Detects event order hazards involving simultaneous reading and writing of the same register in concurrently executing processes. Optional. You must also specify this argument when you simulate the design with `vsim`. Refer to "[Hazard Detection](#)" for more details.

Note



Enabling `-hazards` implicitly enables the `-compat` argument. As a result, using this argument may affect your simulation results.

- `-incr` | `-noincr`

Defines whether vopt reuses design elements or not.

`-incr` — Instructs vopt to only optimize design elements that have changed since a previous optimization. Default

`-noincr` — Instructs vopt to optimize all design elements, even if they have not changed since a previous optimization.

- `+initmem[=<spec>][+{0 | 1 | X | Z}][+<selection>[.]]`

Enables the initialization of memories. Optional.

`=<spec>` — (optional) identifies the types to be initialized.

If you do not specify this option, vlog initializes fixed-size arrays of all these types, where fixed-size arrays may have any number of packed or unpacked dimensions.

`<spec>` can be one or more of the following:

`r` — register/logic, integer, or time types (four-state integral types).

`b` — bit, int, shortint, longint, or byte types (two-state integral types).

`e` — enum types.

You must also add the enum's base type to the initialization specification. If you choose static initialization for an enum type variable with value 0, 1, X, or Z, the simulator assigns that value to the variable, whether it is a valid value or not. If you choose random initialization for an enum type variable, the simulator

generates a random number and uses the (random_number % num_valid_enum_values)th entry of the enum literals to initialize it.

u — sequential UDPs.

+{0 | 1 | X | Z} — (optional) specifies the value to use in initialization for all bits of a memory. For two-state datatypes, X and Z will map to 0.

If you do not specify this option you are preparing the design unit for randomization with vsim +initmem +<seed>.

+<selection>[.] — specifies a design unit name (module, package, interface, or program), or an instance name. An optional trailing '.' after the "selection" means that the initialization recursively descends into the hierarchy.

If you do not specify this argument, the initialization is provided to the entire design.

If initialization is recursively extended to descendants of a given design unit, and a descendant has an explicit initialization specification applied to it, the child's initialization specification overrides the parent's initialization specification. The initialization specifications of the parent and child are not merged together (as is done with +acc options). The default initialization state for top modules is "no_init", provided a top module does not have an explicit initialization specification applied to it.

This argument initializes static variables in any scope (package, \$unit, module, interface, generate, program, task, function). However, it does not affect:

- automatic variables
- dynamic variables
- members of dynamic variables
- artificially generated variables, such as #randstate#

Because you can specify +initmem on the vlog and vopt command line, the priority of the specifications are as follows:

- 1) vopt ... +initmem+1+top.foo
- 2) vlog ... +initmem+0
- 3) vopt ... +initmem+Z

This argument will not override any variable declaration assignment, such as:

```
reg r = 1'b0
```

- +initreg[=<spec>][+{0 | 1 | X | Z}][+<selection>[.]]

Enables you to initialize registers. Optional.

=<spec> — (optional) identifies the types to be initialized.

If you do not specify this option, vlog initializes variables of all these types.

<spec> can be one or more of the following:

r — register/logic, integer, or time types (four-state integral types).

Notifier registers are not initialized by the +initreg option.

b — bit, int, shortint, longint, or byte types (two-state integral types).

e — enum types.

You must also add the enum's base type to the initialization specification. If you choose static initialization for an enum type variable with value 0, 1, X, or Z, the simulator assigns that value to the variable, whether it is a valid value or not. If you choose random initialization for an enum type variable, the simulator generates a random number and uses the (random_number % num_valid_enum_values)th entry of the enum literals to initialize it.

u — sequential UDPs.

If a sequential UDP contains an "initial" statement, that initial value overrides all +initreg-related functionality. For other sequential UDPs, the +initreg option takes effect as described for regular variables. In case a sequential UDP does not contain an "initial" statement, and it wasn't compiled with +initreg in effect, the UDPs initial value will be taken from its instantiating parent scope (provided that scope has +initreg options in effect).

+{0 | 1 | X | Z} — (optional) specifies the value to use in initialization. For two-state datatypes, X and Z will map to 0.

If you do not specify this option you are preparing the design unit for randomization with vsim +initreg +<seed>

+<selection>[.] — specifies a design unit name (module, package, interface, or program), or an instance name. An optional trailing '.' after the "selection" means that the initialization recursively descends into the hierarchy.

If you do not specify this argument, the initialization is provided to the entire design.

If initialization is recursively extended to descendants of a given design unit, and a descendant has an explicit initialization specification applied to it, the child's initialization specification overrides the parent's initialization specification. The initialization specifications of the parent and child are not merged together (as is done with +acc options). The default initialization state for top modules is "no_init", provided a top module does not have an explicit initialization specification applied to it.

This argument initializes static variables in any scope (package, \$unit, module, interface, generate, program, task, function). However, it does not affect:

- automatic variables
- dynamic variables
- members of dynamic variables
- artificially generated variables, such as #randstate#

Because you can specify +initreg on the vlog and vopt command line, the priority of the specifications are as follows:

```
1) vopt ... +initreg+1+top.foo
```

```
2) vlog ... +initreg+0
3) vopt ... +initreg+Z
```

This argument will not override any variable declaration assignment, such as:

```
reg r = 1'b0
```

- **-ka**

(optional) This switch, which is short for “keep alternate”, informs vopt that it should attempt to preserve a 32-bit version of an optimized design, if it exists, if compiling with a 64-bit version of vopt (and vice versa).

When you specify **-ka**, vopt attempts to retain the 32- or 64-bit version of the design unit, thus overlaying both into the same optimized design. This allows you to maintain a single optimized design unit that can be simulated by both 32- and 64-bit versions of the simulator. This switch requires you to follow these compatibility rules:

- All designs that go into the optimized design must be the same, specifically nothing can be changed or recompiled.
- You must use the same version of the simulator for both compilations.
- All command line and modelsim.ini options that affect compilation must be the same, including library mapping.

For example, if you are using a 32-bit version of vopt and enter:

```
vopt -o mydesign top1 top2 +acc
```

then at another point use a 64-bit version of vopt:

```
vopt -o mydesign top1 top2 +acc -ka
```

“mydesign” will result in having both 32- and 64-bit versions, assuming you did not change any modelsim.ini settings or recompile any design units.

This command is most useful for users who distribute black-boxed design units, specifically if the recipient switches between 32- and 64-bit execution.

- **-keep_delta**

Disables optimizations that remove delta delays. Optional.

Delta delays result from zero delay events. Those events are normally processed in the next iteration or "delta" of the current timestep. **Vopt** implements optimizations that can remove delta delays and process an event earlier.

- **-L <libname>**

Searches the specified resource library for precompiled modules. The library search options you specify here must also be specified when you run the [vsim](#) command. Optional.

- **-Lf <libname>**
Same as **-L**, but the specified library is searched before any **'uselib** directives. (Refer to “[Library Usage](#)” and “[Verilog-XL Compatible Compiler Arguments](#)” for more information.) Optional.
- **-lowercasepslpragma**
Forces the optimization utility to accept only lower case (embedded) PSL pragmas.
- **+maxdelays**
Selects maximum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by using **vsim -sdfmax**.
- **+mindelays**
Selects minimum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by using **vsim -sdfmin**.
- **-modelsimini <ini_filepath>**
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-mti_trace_vlog_calls**
Enables viewing of SystemVerilog class contents in the Wave window. Optional.
- **-nocover**
Disables code coverage on all source files, regardless of any **-cover** arguments specified to [vlog](#) or [vopt](#). Optional.
- **-nocovercells**
Disables code coverage of modules defined by **'celldefine** and **'endcelldefine** compiler directives, or compiled with the **-v** or **-y** arguments. Optional. Can be used to override the [CoverCells](#) compiler control variable in the *modelsim.ini* file.
- **-nocoverfec**
Prevents focused expression coverage (FEC) from being enabled for coverage collection. By default, both UDP and FEC coverage statistics are enabled for collection. You can customize the default behavior with the [CoverFEC](#) variable in the *modelsim.ini* file. Optional.
- **-nocoverudp**
Prevents UDP coverage from being enabled for coverage collection. By default, both UDP and FEC coverage statistics are enabled for collection. You can customize the default behavior with the [CoverUDP](#) variable in the *modelsim.ini* file. Optional.

- `-nodebug[=ports | =pli | =ports+pli]`

Hides, within the GUI and other parts of the tool, the internal data of all compiled design units. Optional.

`-nodebug` — The switch, specified in this form, does not hide ports, due to the fact that the port information may be required for instantiation in a parent scope.

The design units' source code, internal structure, registers, nets, etc. will not display in the GUI. In addition, none of the hidden objects may be accessed through the Dataflow window or with commands. This also means that you cannot set breakpoints or single step within this code. It is advised that you not compile with this switch until you are done debugging.

Note that this is not a speed switch like the “nodebug” option on many other products. Use the `vopt` command to increase simulation speed.

`-nodebug=ports` — additionally hides the ports for the lower levels of your design; it should be used only to compile the lower levels of the design. If you hide the ports of the top level you will not be able to simulate the design.

Do not use the switch in this form when the parent is part of a `vopt -bbox` flow or for mixed language designs, especially for Verilog modules to be instantiated inside VHDL.

`-nodebug=pli` — additionally prevents the use of pli functions to interrogate individual modules for information.

You should be aware that this form will leave a "nodebug" module untraversable by PLI.

`-nodebug=ports+pli` — you can combine the behavior of `=ports` and `=pli` in this manner.

This functionality encrypts entire files. The ``protect` compiler directive allows you to encrypt regions within a file.

- `-nofsmresettrans`

Disables recognition of implicit asynchronous reset transitions. Optional. This has the effect of excluding asynchronous reset transitions from any coverage results.

- `-nofsmxassign`

Disables recognition of FSMs containing x assignment. Optional.

- `+nolibcell`

By default all modules compiled from a source library are treated as though they contain a ``celldefine` compiler directive. This option disables this default. The ``celldefine` directive only affects the PLI access routines `acc_next_cell` and `acc_next_cell_load`. Optional.

- `-nologo`

Disables the startup banner. Optional.

- +nosparse[+<selection> [.]]

Identifies which memories are considered "not sparse", which instructs the tool to override the rules for allocating storage for memory elements only when necessary.

If you use +nosparse on a given memory, the tool will simulate the memory normally. Refer to [Sparse Memory Modeling](#) for more information.

<selection> — enables access for specific Verilog design units, scopes, or design objects (vars, mem). Multiple selections are allowed, with each separated by a "+" (+nosparse=+top1+top2). If no selection is specified, then all modules are affected. You can use a path delimiter to select unique instances or objects (+nosparse=+/top/ul.). If you specify a module name (+nosparse=+Demux), pertinent objects inside the module are selected.

. — indicates the selection occurs recursively downward from the specified module or instance

- +nospecify

Disables specify path delays and timing checks. Optional.

- -note <msg_number>[,<msg_number>,...]

Changes the severity level of the specified message(s) to "note." Optional. Edit the [note](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

- +notimingchecks

Removes all timing check entries from the design as it is parsed. Optional. To disable checks on individual instances, use the [tcheck_set](#) command.

Specifying **vopt +notimingchecks** or -GTimingChecks=<FALSE/TRUE> will fix the TimingChecksOn generic value in VITAL models to FALSE for simulation. As a consequence, using **vsim +notimingchecks** at simulation may not have any effect on the simulation depending on the optimization of the model.

- +nowarn<CODE>

Disables warning messages in the category specified by <CODE>. Optional. Warnings that can be disabled include the <CODE> name in square brackets in the warning message. For example,

```
** Warning: test.v(15): [RDGN] - Redundant digits in numeric literal.
```

This warning message can be disabled by specifying **+nowarnRDGN**.

- -nowarn <category_number>

Selectively disables a category of warning message. Optional. Multiple **-nowarn** switches are allowed. Warnings may be disabled for all compiles via the Main window **Compile > Compile Options** menu command or the *modelsim.ini* file (refer to [modelsim.ini Variables](#)).

The warning message categories are described in [Table 2-12](#):

Table 2-12. Warning Message Categories for vopt -nowarn

Category number	Description
1	unbound component
2	process without a wait statement
3	null range
4	no space in time literal
5	multiple drivers on unresolved signal
6	VITAL compliance checks
7	VITAL optimization messages
8	lint checks
9	signal value dependency at elaboration
10	VHDL-1993 constructs in VHDL-1987 code
11	PSL warnings
12	non-LRM compliance in order to match Cadence behavior
13	constructs that coverage can't handle
14	locally static error deferred until simulation run
15	SystemVerilog assertions using local variable

- `+num_opt_cell_conds+<value>`

Restricts gate level optimization capacity for accepting cells with I/O path and timing check conditions. Optional.

value — integer between 32 and 1023, inclusive. where the default value is 1023.

- `-O0 | -O1 | -O4 | -O5`

Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

Please refer to the section "[Optimizing Designs with vopt](#)" in the User's Manual for detailed information on using vopt to perform optimization.

- Enable some optimization with **-O1**. Optional.
- Enable most optimizations with **-O4**. Default.

- Enable maximum optimization with **-O5**. Optional. **-O5** attempts to optimize loops and prevents variable assignments in situations where a variable is assigned but is not actually used.
- **-ocf <filename>**
Specifies an OCF (.ocf) or OCM (.ocm) file that controls the visibility for the simulation, based on using the learn flow. Refer to the section “[Description of Learn Flow Control Files](#)” in the User’s Manual for more information.
- **-pa_all**
Writes power elements and register/latch information to the output file for Power Aware verification.
- **-pa_allowzerotimeevent**
Enables corruption, isolation, or release of signals at time 0.
- **-pa_bbox <dut_name>**
Change to a different device-under-test when running simulation, where `dut_name` is the name of a Verilog or VHDL design element under test. If you are using a library that is not “work,” this name must be of the following form:

```
library_name.dut_name
```


When you use this argument for the first time on the specified DUT, you also need to run `vsim -pa_bboxprefix`. The result is that ModelSim creates Power Aware dumps that you can change during simulation runtime without having to rerun `vopt`.
- **-pa_behavlogfile <filename>**
Generates a list of all the non-synthesizable constructs found in design to a text file, specified by `<filename>`. This list identifies all the constructs that must be in always-on power domain.
- **-pa_ce {o | os | osw}**
Overrides the corruption extent specified in the PCF file. Specific to [Power Aware Verification](#).
 - o — sets the corruption extent to outputs only. Equivalent to `-pa_out`.
 - os — sets the corruption extent to outputs and sequential elements. Equivalent to `-pa_all`.
 - osw — sets the corruption extent to outputs and sequential and non-sequential wires.
- **-pa_cfg <filename>**
Specifies the power configuration file to be used for the verification.

vopt

- `-pa_checks={rop | rpo | rcs | rsa | r | iep | idp | irc | it | isa | i | t | cp | p | a | npu | ugc | upc | s | sdl | sil | sml | snl | srl | svl | ul | uil | uml}`

Note

To specify more than one checking value, use the + operator between values.
For example:

```
vopt -pa_checks=rop+cp+a
```

- rop** — States that there were retention signals mapped to the power domain. However, these signals were not asserted when the power was switched off.
- rpo** — States that there was an error in the sequence of triggering of the retention and power signal. For retention to succeed, the power should be high. However, this check is triggered when that is not the case
- rcs** — Certain Power Aware models require that the clock/latch enable must be at a certain value when retention takes place. This check displays error when this condition is not satisfied. If latch is enabled and can change its value, triggering retention can potentially cause race in the stored value. This is also a check against such conditions.
- rsa** — Some Power Aware models require that the clock not toggle when the power is down. This check helps in monitoring this condition.
- r** — Enable retention checking performed by all preceding retention arguments.
- iep** — Detects conditions where the isolation signal is not enabled when the power to a power domain goes down.
- idp** — Triggered when the isolation signal is enabled and the power to a power domain has already gone down.
- irc** — It is possible for the isolated ports to change value when the isolation is triggered. This check catches such error conditions at the negative and positive edges of isolation enable. (Not supported for UPF)
- it** — Ensures that the isolated ports do not toggle when isolation is high. (Not supported for UPF)
- isa** — Some isolation strategies require that another reset signal be used to freeze the value of isolation. This check ensures that the isolated value is the same at posedge of reset and at posedge of isolation enable signal. (Not supported for UPF)
- i** — Enables isolation checking performed by all preceding isolation arguments. (For UPF, this value enables only iep and idp.)
- t** — Enables the simulator to catch a condition where inputs to the power domain toggle even when the power domain is turned off.
- cp** — Helps catch conditions when the power signal to any power domain gets corrupted during simulation.
- p** — Identifies when each power domain is switched on or off.

-
- a — Helps determine if all the signals in an always-on power domain are ever corrupted. It is possible that certain blocks of an RTL might be in an always-on power domain; corrupting signals in this domain will result in corrupting other Power Aware blocks.
- npu — Detects whether asynchronous (set/reset) controls for non-retention flip-flops are asserted at power-up.
- ugc — Detects a glitch (multiple value changes at same time step) on controls of power objects (isolation, retention, switches).
- upc — Catches any switching off of retention/isolation supplies during active retention/isolation period.
- s — Enables all static checks for level shifting.
- sdl — Checks for a level shifter that has not been inserted because you have specified `set_level_shifter -no_shift` in the UPF file, where running a static analysis of power domain boundaries results in one of the following:
- Level shifter not required — the report contains a counter labeled “Not inserted” that shows the number of level shifters found but not inserted.
 - Level shifter required — ModelSim has determined a need for a level shifter and flags it as an incorrect level shifter.
- sil — Checks for any level shifter that shifts in a particular direction (such as high-to-low) but the actual potential difference inferred from state table is in a different direction (such as low-to-high).
- sml — Checks for any level shifter that is missing at the particular source to sink power domain.
- snl — Checks for a level shifter present at the boundary where there is no voltage information available. This could be possible due to a missing entry in the Power State Table. ModelSim ignores such level shifters for multi-voltage analysis and flags them as Not Analyzed.
- srl — Checks for any level shifter whose input and output supplies have no voltage difference. Thus the tool flags them as a redundant level shifter.
- svl — Checks for valid level shifters that have matching voltage information and proper direction of shift.
- ul — Enables all dynamic checks for level shifting.
- uil — Checks for incorrect level shifter; During simulation there will be an assertion failure whenever there is a mismatch in the shift direction of the level shifter specified at the domain boundary.
- uml — Checks for missing level shifter; During simulation there will be an assertion failure whenever there is a difference between the operating voltage of source and sink domains.

- `-pa_checkseq="<t1> [<unit>] [...<tn>]"`

Turns assertion (all dynamic checking) on and off at specified times. This helps remove unnecessary assertion failure messages for a period of time you are not interested in verifying.

<t1> — Any positive integer. Note that *t1* turns checking on.

<tn> — Any positive integer greater than *<t1>* (each *tn* must be greater than *tn-1*). Note that *t1* turns checking on; subsequent values of *tn* alternate turning checking off and on.

<unit> — Unit of simulation time at which you want to turn checking on or off. If you do not specify a unit, the default simulation time multiplier is used.

- `-pa_crptlogfile <filename>`

Creates a log file containing hierarchy of signals corrupted during simulation.

- `-pa_dbgstatic msk | rsn`

Provides debugging on static checking for level shifters. The following values determine the debugging behavior performed:

msk — Captures the connection mask of the signal taken in a particular path. For example, if there is a vector taking different path, the phrase "connected mask" provides information about which bits are used in connecting the particular source and sink.

rsn — Appends information at the end of a report following the phrase "Possible reason" to help pinpoint the cause of the check.

- `-pa_dumpupf <filename>`

Saves all UPF commands to an output file without any processing (even if the commands are not supported by ModelSim). This is an alternate method of using the UPF command, `save_upf`, in uninterpreted mode.

- `-pa_excludefile <filename> [-pa_enhexclude]`

Case 1 — Using `vopt -pa_prefix -pa_replacetop`:

Skips Power Aware processing for a module, its instances in a particular hierarchical path, or signals in the design. The names and locations of these modules, instances, or signals are contained in an exclude file that directs Power Aware processing to skip all instantiation of *<module_name>* found within the instance *<hier_path>*.

<filename> — The name of a text file that specifies modules, instances, or signals you want to exclude from Power Aware verification. You can use the pound sign (#) to begin a comment line. Each entry in the file must be of the following form:

```
<module_name> [-a] <hier_path>
```

where

module_name can be any regular expression (enclosed in quotation marks)

-a is an optional switch that enables recursive exclusion of *module_name*.

hier_path is the full pathname to the instance of the module you want to exclude. When a module instance is skipped, QuestaSim displays the following message:

```
** Note: (vopt-9691) Excluding power aware module '<module_name>'
in path '<hier_path>' .
```

When used with -a, hier_path limits the recursive exclusion to a particular scope.

Case 2 — Using vopt -pa_bbox:

As in Case 1, skips Power Aware processing for a module, its instances in a particular hierarchical path, or signals in the design by using an exclude file. The exclude file specifies instantiation of <module_name> found within the instance <hier_path> in the same way as in Case 1, with the following exception:

hier_path specifies the path to the instance of the module you want to exclude beginning at or below the instance of the pa_bbox dut_name (instead of using the full pathname)

-pa_enhexclude — Excludes a signal or an instance in a given module in a particular hierarchical path in the design (signal names and instance path names can be a regular expression enclosed in quotation marks). To exclude a design element in the whole hierarchy, do not specify an instance path name. For signal or instance exclusion, each entry in the exclude file must be of the following form:

```
<module_name> [instance_pathname] [[-s | -r] signal_name {, signal_name...}]
```

where -s specifies signal exclusion, -r specifies recursion (applied to s, where all occurrences inside only that module scope will be excluded).

- -pa_genrpt=[{nv | v} [{us | ud | u}] [b]]

Note



To specify more than one report value, use the + operator between values. For example:

```
vopt -pa_genrpt=v+us+b
```

Generates a Power Aware report file that is saved to the current directory. To change the name of the report file (default: report.mspa.txt), use the -pa_reportfile argument. Use one of the following values to specify the level of information to include in this file:

nv — (non-verbose) Displays a count of the power aware elements in the design with respect to the power domains in the PCF file. This is the default if you do not specify nv or v.

v — (verbose) Displays the hierarchical path of individual power aware elements as well as their counts.

Use one of the following values to generate a report on the contents of the UPF:

ud — Dynamic reporting. Includes time and polarity of controls (such as control port of a power switch, retention save and restore signals, isolation enable signal), plus

Power Domain Status in the form of <Power Domain name>, <Strategy name>, <Control signal names>, <Active Sense>, <Current polarity value>. Dynamic UPF reporting is displayed in the transcript window.

us — Static reporting. Includes power domains, switches (with control ports and states), retention strategies (with save and restore controls), and isolation strategies (with isolation control and a list of isolated signals). Static UPF information is sent to a text file named report.upf.txt.

u — Static and dynamic.

Use the following value to modify the contents of report.upf.txt:

b — writes bitwise expanded information for isolated and level shifted ports into report.upf.txt.

Using the `-pa_genrpt` argument generates the following text files:

- **report.alwaysontxt** — Signals are in the always-on power domain. They should never get corrupted due to any power activity.

Example excerpt:

```
padd3 sub_total ( /top/bot2 )  
  
NPM_FF # 3  
  /top/bot2/out 3  
  
OUTPUT # 1  
  /top/bot2/clock 1
```

- **report.corruption.txt** — Signals that are in a power domain but not in always-on power domain (that is, they are corrupted during simulation).

Example excerpt:

```
padd4 sub_total ( /top/bot3 )  
NPM_FF # 3  
  /top/bot3/out 3  
  
OUTPUT # 1  
  /top/bot3/clock 1
```

- **report.isolation.txt** — All isolation cells specified with your PCF or UPF.

Example excerpt:

```
ISO_id: iso_bot2 , extent: /tb/top , enable: (/tb/isol) , reset:  
!(/tb/isol)  
ports:  
  /tb/top/output2  
  /tb/top/in_sl
```

- report.upf.txt — Static UPF report. (Dynamic UPF reporting is displayed in the transcript window.)
- -pa_gls
Enables gate-level simulation for a Power Aware analysis.
- -pa_hiersep <alphanum_character>
Specifies that the PCF uses a hierarchical path separator different from the default character, which is the slash (/) character.
- -pa_intcrptval0
Sets the default corruption value to 0 (zero) instead of 'left.
- -pa_libertylibs=<database_pathname>
Specifies the location of a liberty database for use with gate-level simulation.
- -pa_lib <library_pathname>
Specifies a different library location for Power Aware verification results than the current work directory. This is required if you want to keep results on the same design for different modes or if you are using gate-level simulation. You can use different libraries to preserve all sets of results, and then invoke vsim with the -pa_lib argument to specify which library location you want to use.
- -pa_lsthreshold <real>
Sets a global threshold level for a Power Aware analysis containing multiple voltage levels, where <real> is any numerical value that specifies a voltage threshold.

Use this argument when you know that level shifting is not required for particular range of voltage differences—you can then specify a global threshold. Otherwise, ModelSim will flag missing level shifter errors even if the potential difference between two domains is within an acceptable range.
- -pa_modeltype=2
Directs ModelSim to ignore invalid transitions (such as X->1 or 0->X) on power-control signals for Power Aware simulation. This allows you to restrict Power Aware models to allow only valid transitions such as 0->1.
- -pa_nopcfctrlcheck
Prevents the default check for existence of the power control statement (PCS) signals. This avoids reporting an error condition (Signal Not Found) when the PCS is not in the hierarchy within which **vopt** is invoked.
- -pa_opt={0 | 1 | 2 | 6 | 7}
Specifies how ModelSim detects non-retention flip-flops.
 - 0 — No detection or optimization of non-retention flip-flops.
 - 1 — Detects asynchronous set/reset of non-retention flip-flops at power-up.

2 — Detects and optimizes asynchronous set/reset of non-retention flip-flops at power-up. Default.

6 — Optimizes instances in the design that exhibit retention capability.

7 — Optimizes instances in the design that do not exhibit retention capability.

- **-pa_out**

Writes only power elements to the output file.

- **-pa_prefix <hier_path>**

Specifies a hierarchical path to be prepended to path information in the PCF file. The paths in the PCF file start from a specific portion of the design, so you can use this argument to create a full path based on the top module on which you will run the vsim command.

- **-pa_replacetop <string>**

Specifies a different name for the top-level module name and its instance name in the test bench (must be used with **-pa_prefix**). The value you specify for **<string>** replaces the first token specified in the paths defined in the PCF before adding the **pa_prefix <hier_path>** to the paths. However, **<string>** will replace the first token only if the first token in the path is a top-level name that you included when invoking **vopt**. This is because control signals are allowed from a different hierarchy than the DUT itself.

- **-pa_reportfile <filename>**

Specifies an alternate filename to the power-aware report file. The default name is **report.mspa.txt** and it is saved in the current directory. Note that this argument does not create the power-aware report file—you must specify **-pa_genrpt** to produce the report.

- **-pa_rtlinfo**

Run Power Aware verification in verbose mode. This helps identify register/latch modeling errors, such as non-synthesizable clocking styles.

- **-pa_tclfile <filename>**

Use a Tcl file containing Unified Power Format (UPF) commands with **save_upf -u** (a UPF command). Note that **save_upf -u** is not defined by the standard for UPF 1.0, so you must use the **vopt -pa_tclfile** argument to specify the output file when you use **save_upf -u**.

- **-pa_upf <filename>**

Specifies the name of a UPF file to be used by **vopt**.

- **-pa_upfscopeconns**

Activates any **create_upf2hdl_vct** or **create_hdl2upf_vct** commands in the UPF file.

- **-pa_upfsyntaxchecks**

Check only the syntax of UPF file specified by **-pa_upf <filename>**. Note that you must also use the **-pa_upf** argument with this argument.

- **-permissive**
Allows messages in the LRM group of error messages to be downgraded to a warning. Optional.
- **-pslxt**
Enables PSL LTL and OBE operators. Optional. These operators are disabled by default.
- **-pslfile_vh <filename>**
Identifies a VHDL PSL file to be read in during optimization. Optional. PSL files can also be specified during compilation using the [vcom](#) command. Refer to “[Using PSL Assertions in the Optimized](#)” for more information.
- **-pslfile_vl <filename>**
Identifies a Verilog PSL file to be read in during optimization. Optional. PSL files can also be specified during compilation using the [vlog](#) command. Refer to “[Using PSL Assertions in the Optimized](#)” for more information.
- **-quiet**
Disables 'Loading' messages. Optional.
- **-source**
Displays the associated line of source code before each error message that is generated during compilation. Optional; by default, only the error message is displayed.
- **-sc_arg <string> ...**
Specifies a string representing a startup argument which is subsequently accessible from within SystemC via the `sc_argc()` and `sc_argv()` functions (refer to “[Accessing Command-Line Arguments](#)”).

If multiple SystemC startup arguments are specified, each must have a separate **-sc_arg** argument. SystemC startup arguments returned via `sc_argv()` are in the order in which they appear on the command line. White space within the `<string>` will not be treated specially, and the string, white space and all, will be accessible as a single string among the strings returned by `sc_argv()`.
- **-sclib <library>**
Specifies the design library where the SystemC shared library is created. By default, the SystemC shared library is created in the logical work library. This argument is only necessary when the shared library is compiled in a design library other than the logical work directory. See the [sccom](#) command for more information.
- **-sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=<instance>]<sdf_filename>**
(optional) Annotates VITAL or Verilog cells in the specified SDF file (a Standard Delay Format file) with minimum, typical, or maximum timing.

@<delayScale> — scales all values by the specified value. For example, if you specify `-sdfmax@1.5`, all maximum values in the SDF file are scaled to 150% of their original value.

Do not use this option if you scaled the SDF file while using the [sdfcom](#) command.

<instance>= — specifies a specific instance for the associated SDF file. Use this when not performing backannotation at the top level.

<sdf_filename> — specifies the file containing the SDF information.

Note



The tool will issue an error if you specify this switch and your SDF is being annotated to VHDL because the tool does not support compiled SDF for VHDL.

- -sdfmaxerrors <n>

Controls the number of Verilog SDF missing instance messages that will be emitted before terminating vsim. Optional. <n> is the maximum number of missing instance error messages to be emitted. The default number is 5.

Note



The tool will issue an error if you specify this switch and your SDF is being annotated to VHDL because the tool does not support compiled SDF for VHDL.

- +staticchecks[=<args>]

Performs a series of static checks on VHDL and Verilog designs, where <args> is a list of arguments. The command performs all checks if you do not specify any arguments to this switch.

Produces a file (*static_checks.txt*) in the current run directory containing messages about any static check violations. You can rename the file with the -staticchecksfile switch.

r — checks for race conditions, either write-write races (multiple drivers) or read-write races. Resulting messages have the label: `STATIC_RACE_CHECK`.

These checks are only on a per-module basis.

To enable multiple driver checks for VHDL you must also specify the -staticchecksmdvhdl switch. This is because the multiple driver may result from a resolution function.

c — checks for simulation-synthesis mismatches related to full/parallel case pragmas. Resulting messages have the label: `STATIC_CASE_CHECK`.

s — checks for simulation-synthesis mismatches related to sensitivity lists; missing objects, duplicate elements, and redundant elements. Resulting messages have the label: `STATIC_SENSLIST_CHECK`.

f — checks for simulation-synthesis mismatches related to subprograms (functions). Resulting messages have the label: `STATIC_FUNCTION_CHECK`.

x — checks for simulation-synthesis mismatches related to reading of four-state values (X, Z, U, or W). Resulting messages have the label: `STATIC_XZUW_CHECK`.

This check does not issue any warnings for casex reading "x" or "z", or casez reading "z".

d — checks for non-synthesizable constructs; untested edge triggers, implicit state machine with different clocks or where the first statement is not within event control, output driven by multiple clocks, improper mixing of control signals, blocking and non-blocking assignments for the same signal, and asynchronous loading. Resulting messages have the label: `STATIC_SYNTH_CHECK`.

Usage examples:

```
+staticchecks           #Executes all checks
+staticchecks=r        #Executes only the check for race conditions
+staticchecks=rCS      #Executes three of the checks
```

- `-staticchecksfile <filename>`

Redirects the output of the `+staticchecks` switch to `<filename>`. By default, this file is named `static_checks.txt`.

- `-staticchecksmdvhdl`

When used with `+staticchecks=r`, enables the check on multiple driver race conditions for VHDL designs. By default, this check is deactivated because the multiple driver for VHDL could be related to a resolution function.

- `-suppress <msg_number>[,<msg_number>,...]`

Prevents the specified message(s) from displaying. Optional. You cannot suppress Fatal or Internal messages. Edit the `suppress` variable in the `modelsim.ini` file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

- `-tab <tabfile>`

Specifies the location of a Synopsys VCS table file (`.tab`), which `vopt` uses to improve the visibility of PLI functions in the design.

`<tabfile>` — The location of a `.tab` file containing information about PLI functions. The tool expects the `.tab` file to be based on Synopsys VCS version 7.2 syntax. Because the format for this file is non-standard, changes to the format are outside of the control of Mentor Graphics.

When you use the Three-step optimization flow, you must specify this switch on both the `vopt` and `vsim` command lines. This is because `vopt` uses this file to improve the visibility of PLI functions and `vsim` uses it to register the PLI functions.

- `-time`

Reports the "wall clock time" `vopt` takes to optimize the design. Optional. Note that if many processes are running on the same system, wall clock time may differ greatly from the actual "cpu time" spent on `vopt`.

- `-timescale <time_units>/<time_precision>`

Specifies the default timescale for modules not having an explicit timescale directive in effect during compilation. Optional. The format of the `-timescale` argument is the same as that of the ``timescale` directive. The format for `<time_units>` and `<time_precision>` is `<n><units>`. The value of `<n>` must be 1, 10, or 100. The value of `<units>` must be fs, ps,

ns, us, ms, or s. In addition, the `<time_units>` must be greater than or equal to the `<time_precision>`.

- `+typdelays`
Selects typical delays from the "min:typ:max" expressions. Default. If preferred, you can defer delay selection until simulation time by using **`vsim -sdfmin`**.
- `-version`
Returns the version of the optimizer as used by the licensing tools. Optional.
- `-warning <msg_number>[,<msg_number>,...]`
Changes the severity level of the specified message(s) to "warning." Optional. Edit the [warning](#) variable in the `modelsim.ini` file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- `-work <library_name>`
Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional. By default, the optimized output for the design is added to the **work** library. The specified pathname overrides the pathname specified for work in the project file.
- `<design_unit>`
One or more top-level design units that you want to optimize. Required.
- `-o <name>`
Specifies a name for the optimized version of the design. Required. The name can contain lower- and upper-case alpha characters, numeric characters or underscores (`_`).

Examples

- Run optimizations on top-level design unit *top* and produce an optimized design unit named "mydesign". The simulator **`vsim`** is then invoked on design unit *mydesign*.

```
vopt top -o mydesign
vsim mydesign
```

- Run optimizations both top-level design units *top* and *testtop* and produce a global optimized design unit named mydesign.

```
vopt top testtop -o mydesign
```

- Run optimizations on top-level design unit *top* but preserve all visibility. Names the optimized design "mydesign."

```
vopt top +acc -o mydesign
```

- Run optimizations on top-level design unit *top* but preserve visibility on sub-module *foo*. Names the optimized design "mydesign."

```
vopt top +acc+foo -o mydesign
```


- Run optimizations on top-level design unit *top* but preserve visibility on sub-module *foo* and all its children.

```
vopt top +acc+foo. -o mydesign
```

- Run optimizations on top-level design unit *top* but enable net and register access in all modules in the design. Names the optimized design "mydesign."

```
vopt top +acc=rn -o mydesign
```

- Run Power Aware verification with the following specifications:

testbench = tb

design-under-test (DUT) = top

DUT instance name within testbench = inst

power control statement (PCS) = POWER pdd /top, /top/PCS

PCS located within top, as /top/PCS

Invoke vopt with top, as follows:

```
vopt -o topdu top -pa_cfg config_file -pa_all -pa_prefix "/tb/"  
-pa_replacetop "inst"
```

Result: Paths generated in mspa._vopt.v as:

/tb/inst/i1

/tb/inst/PCS

- Run Power Aware verification with the following specifications (changes to PCS):

testbench = tb

design-under-test (DUT) = top

DUT instance name within testbench = inst

PCS located within tb, as /tb/PCS (turn off checking for PCS)

power control statement = POWER pdd /top, PCS

Invoke vopt with top, as follows:

```
vopt -o topdu top -pa_cfg config_file -pa_all -pa_prefix "/tb/"  
-pa_replacetop "inst" -pa_nopcfctrlcheck
```

Result: Paths generated in mspa._vopt.v as:

/tb/inst/i1

/tb/PCS

- Run Power Aware verification with the following specifications (changes to PCS):

testbench = tb

design-under-test (DUT) = top

DUT instance name within testbench = inst

PCS located within controller within tb, as /tb/controller/PCS

(turn off checking for PCS)

power control statement = POWER pdd /top, controller/PCS

Invoke vopt with top, as follows:

```
vopt -o topdu top -pa_cfg config_file -pa_all -pa_prefix "/tb/"  
-pa_replacetop "inst" -pa_nopcfctrlcheck
```

Result: Paths generated in mspa._vopt.v as:

```
/tb/inst/i1  
/tb/controller/PCS
```

- Run Power Aware verification with the following specifications (changes to DUT):

testbench = tb

design-under-test (DUT) = top

DUT instance name within testbench = inst_top or

power control statement (PCS) = POWER pdd /top, /top/PCS

PCS located within top, as /top/PCS

Invoke vopt with top, in either of the following ways:

```
vopt -o topdu top -pa_cfg config_file -pa_all -pa_prefix "/tb/"  
-pa_replacetop "inst_top"
```

OR

```
vopt -o topdu top -pa_cfg config_file -pa_all -pa_prefix "/tb/inst_"
```

Result: Paths generated in mspa._vopt.v as:

```
/tb/inst_top/i1  
/tb/inst_top/PCS
```

- The -fsmmultitrans option enables detection and reporting of multi-state transitions when used with the +cover f argument.

```
vopt -o opt_ttop top +cover=f -fsmmultitrans
```

- Run Power Aware using -pa_prefix and -pa_replacetop while specifying an exclude file (Case 1—exclude file must specify full hierarchy of excluded instance):

```
vopt -pa_prefix "/interleaver_tester/" -pa_replacetop  
"interleaver1" -o my_chip -pa_excludefile bar_exclude
```

#Contents of bar_exclude:

```
foo /interleaver_tester/interleaver1/foo[1]/bar -s ctl
```

- Run Power Aware using -pa_bbox while specifying an exclude file (Case 2—exclude file needs to specify hierarchy of excluded instance only beginning at dut_name of pa_bbox instance):

```
vopt -pa_bbox "interleaver" -o my_chip -pa_excludefile bar_exclude
```

#Contents of bar_exclude:

```
foo foo[1]/bar -s ctl
```

See also

[“Optimizing Designs with vopt”](#)

vsim

The **vsim** command invokes the VSIM simulator, which you can use to view the results of a previous simulation run (when invoked with the **-view** switch), or to view coverage data stored in the UCDB from a previous simulation run (when invoked with the **-viewcov** switch).

You can simulate a VHDL configuration or an entity/architecture pair, a Verilog module or configuration, a SystemC module, or an optimized design. If you specify a VHDL configuration, it is invalid to specify an architecture. During elaboration **vsim** determines if the source has been modified since the last compile.

To manually interrupt design loading, use the Break key or <Ctrl-C> from a shell.

You can invoke **vsim** from a command prompt or in the Transcript window of the Main window. You can also invoke it from the GUI by selecting Simulate > Start Simulation.

All arguments to the **vsim** command are case sensitive; for example, **-g** and **-G** are not equivalent.

Syntax

Note



This Syntax section presents all of the **vsim** switches in alphabetical order, while the Arguments section groups the arguments into the following sections:

- [Arguments, all languages](#)
- [Arguments, VHDL](#)
- [Arguments, Verilog](#)
- [Arguments, SystemC](#)
- [Arguments, object](#)

vsim [options]

[options]:

- [\[-0in\]](#) [[-0in_options](#) "<options>"] [[-32](#) | [-64](#)]
- [\[-absentisempty\]](#) [[-assertdebug](#)] [[+alt_path_delays](#)] [[-assertfile](#) <filename>] [[-assume](#) | [-noassume](#)] [[-autoexclusionsdisable](#)=<exclusion_type>]
- [\[-c\]](#) [[-capacity](#)] [[-colormap](#) new] [[-compress_elab](#)] [[-coverage](#)] [[-covercountnone](#)] [[-cpppath](#) <filename>] [[-cvg63](#) | [-nocvg63](#)] [[-cvghaltillbin](#)] [[-cvgsingledefaultbin](#)]
- [\[-debugDB](#)=<db_pathname>] [[+delayed_timing_checks](#)] [[-display](#) <display_spec>] [[-displaymsgmode](#) both | tran | wlf] [[-do](#) "<command_string>" | <macro_file_name>] [[-donotcollapsepartiallydriven](#)] [[-dpiexportobj](#) <objfile>] [[-dpioutoftheblue](#) 0 | 1 | 2] [[+dumpports+collapse](#) | [+dumpports+nocollapse](#)] [[+dumpports+direction](#)] [[+dumpports+no_strength_range](#)] [[+dumpports+unique](#)]
- [\[-error](#) <msg_number>[,<msg_number>,...]] [[-elab](#) <filename>] [[-elab_cont](#) <filename>] [[-elab_defer_fli](#)] [[-enumbaseinit](#)] [[-errorfile](#) <filename>] [[-extendedtogglemode](#) 1|2|3]

[-f <filename>] [-fatal <msg_number>[,<msg_number>,...]]
 [-filemap_elab <HDLfilename>=<NEWfilename>] [-foreign <attribute>]
 [-g<Name>=<Value> ...] [-G<Name>=<Value> ...] [-gblso <filename>]
 [-geometry <geometry_spec>] [-gui]
 [-hazards] [-help]
 [-i] [+initmem+<seed>] [+initreg+<seed>] [-installcolormap]
 [-keeploaded] [-keeploadedrestart] [-keepstdout]
 [-l <filename>] [-lib <libname>] [<license_option>] [-L <library_name> ...]
 [-learn <root_file_name>] [-Lf <library_name> ...] [-L mtiPA]
 [<library_name>.<design_unit>] [-load_elab <filename>]
 [+maxdelays] [+mindelays] [-memprof] [-memprof+call] [-memprof+file=<filename>]
 [-memprof+fileonly=<filename>] [-modelsimini <ini_filepath>]
 [-msgmode both | tran | wlf] [-multisource_delay min | max | latest]
 [+multisource_int_delays]
 [-name <name>] [+no_autdct] [-noautoldlibpath] [-nodpiexports [-dpiexportcheckref]]
 [-no_autoacc]
 [+no_cancelled_e_msg] [+no_glitch_msg] [+no_neg_tchk] [+no_notifier]
 [+no_path_edge] [+no_pulse_msg] [-no_risefall_delaynets]
 [+no_show_cancelled_e] [+no_tchk_msg] [-nocollapse] [-nocapacity] [-nocompress]
 [-nocrossautobins] [-noexcludehiz] [-nofileshare] [-noimmedca]
 [-togglvlogints | -notogglvlogints] [-noglitch] [+nosdferror] [+nosdfwarn]
 [+nospecify] [-notoggleints] [-note <msg_number>[,<msg_number>,...]]
 [+notimingchecks] [+nowarnBSOB] [+nowarn<CODE>] [+ntc_warn]
 [-oldvhdlforgennames] [-onfinish ask | stop | exit] [-OVMdebug]
 [-pa] [-pa_allowzerotimeevent] [-pa_bboxprefix= <pathname>]
 [-pa_gls <testbench_top>] [-pa_lib <pathname>] [-pa_zcorrupt] [-pli "<object list>"]
 [-permissive] [-plicompatdefault [latest | 2005 | 2001]] [+<plusarg>] [-printsimstats]
 [-psl | -nopsl] [-psloneattempt]
 [-pslinfinitethreshold=<integer>] [+pulse_e/<percent>] [+pulse_e_style_ondetect]
 [+pulse_e_style_onevent] [+pulse_r/<percent>]
 [-quiet]
 [-restore <filename>] [-runinit]
 [-sc_arg <string> ...] [-scdpidebug] [-sclib <library>] [+sdf_iopath_to_prim_ok]
 [+sdf_nocheck_celltype]
 [-sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=<sdf_filename>]
 [-sdfmaxerrors <n>] [-sdfnoerror] [-sdfnowarn] [+sdf_report_unannotated_insts]
 [+sdf_verbose] [-std_input <filename>] [-std_output <filename>]
 [+show_cancelled_e]
 [-solveengine {auto | bdd | act}] [-solvefaildebug] [-solverev <version>]
 [-strictvital] [-suppress <msg_number>[,<msg_number>,...]] [-sv_lib <shared_obj>]
 [-sv_liblist <filename>] [-sv_root <dirname>] [-sv_seed <integer> | random]
 [-sva | -nosva] [-sync]

```

[-t [<multiplier><time_unit>] [-tab <tabfile>] [-tag <string>] [-title <title>]
[-togglecountlimit <int>] [-togglefixedsizearray | -notogglefixedsizearray]
[-togglemaxfixedsizearray <int>] [-togglemaxintvalues <int>]
[-togglemaxrealvalues <int>] [-togglepackedasvec] [-togglevlogenumbits]
[-notoggleints] [-togglevlogints | -notogglevlogints]
[-togglevlogreal | -notogglevlogreal] [-togglewidthlimit <int>]
[-trace_foreign <int>] [+transport_int_delays]
[+transport_path_delays]

[+typdelays]

[-v2k_int_delays] [-vcdread <filename>] [-vcdstim [<instance>=<filename>]
[-version] [-view [<alias_name>=<WLF_filename>]
[-viewcov [<dataset_name>=<UCDB_filename>] [-visual <visual>] [-vital2.2b]
[+vlog_retain_on | +vlog_retain_off]
[+vlog_retain_same2same_on | +vlog_retain_same2same_off]
[-vopt | -novopt] [-voptargs="<args>"] [-vopt_verbose]

[-warning <msg_number>[,<msg_number>,...]] [-wlf <file_name>] [-wlfcachesize
<n>]
[-wlfcollapsedelta] [-wlfcollapsetime] [-wlfcompress] [-wlfdeleteonquit] [-wlflock]
[-wlfnocollapse] [-wlfnocompress] [-wlfnodeleteonquit] [-wlfnoLOCK] [-wlfnoopt]
[-wlfopt] [-wlfsimcachesize <n>] [-wlfslim <size>] [-wlftlim <duration>]
[-wlfthreads | -wlfnothreads]

```

Arguments, all languages

- -0in

Invokes **0-In ccl** from within ModelSim. Optional. For the invocation to work, the environment variable HOME_0IN must be set to the proper pathname.
- -0in_options "<options>"

Passes the specified options to **0-In ccl**. Optional. The options must be enclosed in parentheses. Consult the 0-In documentation for details on the available options. For the invocation to work, the environment variable HOME_0IN must be set to the proper pathname.
- -32 | -64

Specifies whether vsim uses the 32- or 64-bit executable, where -32 is the default.

These options override the MTI_VCO_MODE environment variable, which applies only to executables used from the <install_dir>/bin/ directory. Therefore, these options are ignored if you run vsim from an <install_dir>/<platform>/ directory.

You can only specify these options on the command line, therefore they are not recognized as part of a file used with the -f switch.
- -assertdebug

(optional) If assertion is logged, this option stores the assertion data in the WLF file so assertion passes and failures can be analyzed in the assertion debug pane of the Wave window. It also:

- enables action settings for assertion passes, starts, and antecedent matches;
- enables assertion thread viewing (ATV) feature;
- enables failed expression analysis;
- enables extended count information including: pass count, vacuous pass count, disabled count, attempted count, active thread count, peak active thread count;
- enables causality traceback feature;
- enables reporting of assertion pass messages for PSL only (disabled by default).

The default behavior can be changed by setting the [AssertionDebug](#) variable in the *modelsim.ini* file. If no debuggable assertions are found in the design, a warning is issued. Refer to “[Analyzing Assertion Failures in the Assertion Debug Pane of the Wave Window](#)” for more info.

When this argument is used, "100% coverage" is achieved if an assertion passes non-vacuously and never failed. Without `-assertdebug` set, "100% coverage" is achieved if the assertion never failed.

- `-assertfile <filename>`

Designates an alternative file for recording VHDL/PSL/SVA assertion messages. Optional. An alternate file may also be specified by the [AssertFile](#) *modelsim.ini* variable. By default, assertion messages are output to the file specified by the [TranscriptFile](#) variable in the *modelsim.ini* file (refer to “[Creating a Transcript File](#)”).

- `-assume`

Simulates PSL and SystemVerilog `assume` directives as though they were `assert` directives. Optional. Default. This option may also be specified with the [SimulateAssumeDirectives](#) variable in the *modelsim.ini* file. Refer to “[Processing Assume Directives](#)” for more information.

- `-autoexclusionsdisable=<exclusion_type>`

(optional) Disables automatic code coverage exclusions for:

- FSMs and it's transitions
- VHDL and SystemVerilog immediate and concurrent assertions and their action blocks.

`<exclusion_type>` — A comma-separated list of values that specify the automatic exclusions you wish to disable, where the values are:

`fsm` — disables automatic exclusion of FSMs

`assertions` — disables automatic exclusion of VHDL and SystemVerilog immediate and concurrent assertions.

`none` — equivalent to “`fsm,assertions`”

To change this default behavior, use the [AutoExclusionsDisable](#) variable in the *modelsim.ini* file. If an FSM state is excluded, then all transitions from and to this state are also excluded.

- **-c**
Specifies that the simulator is to be run in command-line mode. Optional. Refer to “[Modes of Operation](#)” for more information.
- **-capacity**
Enables the fine-grain analysis display of memory capacity (coarse-grain analysis is enabled by default). Optional.
- **-colormap new**
Specifies that the window should have a new private colormap instead of using the default colormap for the screen. Optional.
- **-compress_elab**
Compresses an elaboration file when it is created. Optional. Refer to “[Simulating with an Elaboration File](#)” for more information.
- **-coverage**
Enables code coverage statistics collection during simulation. Optional. Important: in order for coverage to be collected and displayed, you must have used `+cover` options during compilation or optimization.
- **-covercountnone**
Disables the default behavior of the simulator to increment the count of all matching rows in condition and expression coverage UDP tables. Optional. Change the default behavior by editing the `CoverCountAll` variable in the `modelsim.ini` file. Please refer to the [Code Coverage](#) chapter in the User’s Manual for more information.
- **-cpppath <filename>**
Specifies the location of a g++ executable other than the default g++ compiler installed with ModelSim. Optional. Overrides the `CppPath` variable in the `modelsim.ini` file.
- **-display <display_spec>**
Specifies the name of the display to use. Optional. Does not apply to Windows platforms.
For example:

```
-display :0
```
- **-displaymsgmode both | tran | wlf**
Controls the transcription of `$display` system task messages to the transcript and/or the Message Viewer. Refer to the section “[Message Viewer Window](#)” in the User’s Manual for more information and the `displaymsgmode` *.ini* file variable.

both — outputs messages to both the transcript and the WLF file.

tran — outputs messages only to the transcript, therefore they are not available in the Message Viewer. Default behavior

wlf — outputs messages only to the WLF file/Message Viewer, therefore they are not available in the transcript.

The display system tasks displayed with this functionality include: \$display, \$strobe, \$monitor, \$write as well as the analogous file I/O tasks that write to STDOUT, such as \$fwrite or \$fdisplay.

- -debugDB=<db_pathname>

Instructs ModelSim to generate database of dataflow connectivity information to be used for post-sim debug in the Dataflow window. Optional. The database pathname should have a .dbg extension. If a database pathname is not specified, ModelSim creates a database file named *vsim.dbg* in the current directory. See [Post-Simulation Debug Flow Details](#).

- -do “<command_string>” | <macro_file_name>

Instructs vsim to use the command(s) specified by <command_string> or the macro file named by <macro_file_name> rather than the startup file specified in the .ini file, if any. Optional. Multiple commands should be separated by semi-colons (;).

- -donotcollapsepartiallydriven

Prevents the collapse of partially driven and undriven output ports during optimization. Prevents incorrect values that can occur when collapsed.

- -dpioutoftheblue 0 | 1 | 2

Instructs **vsim** to allow DPI out-of-the-blue calls from C functions. The C functions must not be declared as import tasks or functions.

0 — Support for DPI out-of-the-blue calls is disabled.

1 — Support for DPI out-of-the-blue calls is enabled, but debugging support is not available.

2 — Support for DPI out-of-the-blue calls is enabled with debugging support for a SystemC thread.

Debugging support for DPI out-of-the-blue calls from a SystemC method requires two **vsim** arguments entered together at the command line: **-dpioutoftheblue 2** and **-scdpidebug**. See **-scdpidebug** for more information.

Related *modelsim.ini* file variable is [DpiOutOfTheBlue](#).

- +dumpports+collapse | +dumpports+nocollapse

Determines whether vectors (VCD id entries) in dumpports output or collapsed or not. Optional. The default behavior is collapsed, and can be changed by setting the [DumpportsCollapse](#) variable in the *modelsim.ini* file.

- +dumpports+direction

Modifies the format of extended VCD files to contain direction information. Optional.

- **+dumpports+no_strength_range**
Ignores strength ranges when resolving driver values for an extended VCD file. Optional. This argument is an extension to the IEEE 1364 specification. Refer to “[Resolving Values](#)” for additional information.
- **+dumpports+unique**
Generates unique VCD variable names for ports in a VCD file even if those ports are connected to the same collapsed net. Optional.
- **-elab <filename>**
Creates an elaboration file for use with **-load_elab**. Optional. Refer to “[Simulating with an Elaboration File](#)” for more information.
- **-elab_cont <filename>**
Creates an elaboration file for use with **-load_elab** and then continues the simulation. Optional.
- **-elab_defer_fli**
Defers the initialization of FLI models until the load of the elaboration file. Use this argument along with **-elab** to create elaboration files for designs with FLI models that don't support checkpoint/restore. Note that FLI models sensitive to design load ordering may still not work correctly even if you use this argument.
- **-enumbaseinit**
Initializes enum variables in SystemVerilog using the default value of the base type instead of the leftmost value. Optional. Edit the EnumBaseInit variable in the *modelsim.ini* file to set a permanent default.
- **-error <msg_number>[,<msg_number>,...]**
Changes the severity level of the specified message(s) to "error." Optional. Edit the **error** variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- **-errorfile <filename>**
Designates an alternative file for recording error messages. Optional. An alternate file may also be specified by the **ErrorFile** *modelsim.ini* variable. By default, error messages are output to the file specified by the **TranscriptFile** variable in the *modelsim.ini* file (refer to “[Creating a Transcript File](#)”).
- **-extendedtogglemode 1|2|3**
Changes the level of support for extended toggles for the simulation. Optional. The levels of support are:
 - 1 — 0L->1H & 1H->0L & any one 'Z' transition (to/from 'Z')
 - 2 — 0L->1H & 1H->0L & one transition to 'Z' & one transition from 'Z'
 - 3 — 0L->1H & 1H->0L & all 'Z' transitions

Edit the [ExtendedToggleMode](#) variable in the *modelsim.ini* file to set a permanent default.

- -f <filename>

Specifies a file with more **vsim** command arguments. Optional. Allows complex argument strings to be reused without retyping.

Refer to the section "[Argument Files](#)" for more information.

- -fatal <msg_number>[,<msg_number>,...]

Changes the severity level of the specified message(s) to "fatal." Optional. Edit the [fatal](#) variable in the *modelsim.ini* file to set a permanent default. Refer to "[Changing Message Severity Level](#)" for more information.

- -filemap_elab <HDLfilename>=<NEWfilename>

Defines a file mapping during **-load_elab** that lets you change the stimulus. Optional. Refer to "[Simulating with an Elaboration File](#)" for more information.

- -g<Name>=<Value> ...

(optional) Assigns a value to all specified VHDL generics and Verilog parameters that have not received explicit values in generic maps, instantiations, or via defparams (such as top-level generics/parameters and generics/parameters that would otherwise receive their default values). No spaces are allowed anywhere in the specification, except within quotes when specifying a string value. Multiple **-g** options are allowed, one for each generic/parameter, specified as a space separated list.

<Name> — Name of a generic/parameter, exactly as it appears in the VHDL source (case is ignored) or Verilog source. Name may be prefixed with a relative or absolute hierarchical path to select generics in an instance-specific manner. For example, specifying **-g/top/u1/tpd=20ns** on the command line would affect only the *tpd* generic on the */top/u1* instance, assigning it a value of 20ns. Specifying **-gu1/tpd=20ns** affects the *tpd* generic on all instances named *u1*. Specifying **-gtpd=20ns** affects all generics named *tpd*.

<Value> — Specifies an appropriate value for the declared data type of a VHDL generic or any legal value for a Verilog parameter. Make sure the value you specify for a VHDL generic is appropriate for VHDL declared data types. Integers are treated as signed values. For example, **-gp=-10** overwrites the parameter *p* with the signed value of -10.

If more than one **-g** option selects a given generic the most explicit specification takes precedence. For example,

```
vsim -g/top/ram/u1/tpd_hl=10ns -gtpd_hl=15ns top
```

This command sets *tpd_hl* to 10ns for the */top/ram/u1* instance. However, all other *tpd_hl* generics on other instances will be set to 15ns.

Limitation: In general, generics/parameters of composite type (arrays and records) cannot be set from the command line. However, you can set string arrays, `std_logic` vectors, and bit vectors if they can be set using a quoted string. For example,

```
-gstrgen="This is a string"  
-gslv="01001110"
```

The quotation marks must make it into vsim as part of the string because the type of the value must be determinable outside of any context. Therefore, when entering this command from a shell, put single quotes (' ') around the string. For example:

```
-gstrgen=' "This is a string" '
```

If working within the ModelSim GUI, you would enter the command as follows:

```
{-gstrgen="This is a string"}
```

You can also enclose the value escaped quotes (\"), for example:

```
-gstrgen=\"This is a string\"
```

- **-G<Name>=<Value> ...**

(optional) Same as **-g** (see above) except that it will also override generics/parameters that received explicit values in generic maps, instantiations, or from defparams. No spaces are allowed anywhere in the specification, except within quotes when specifying a string value. This argument is the only way for you to alter the generic/parameter, such as its length, (other than its value) after the design has been loaded.

<Name> — Name of a generic/parameter, exactly as it appears in the VHDL source (case is ignored) or Verilog source. Name may be prefixed with a relative or absolute hierarchical path to select generics in an instance-specific manner. For example, specifying **-G/top/u1/tpd=20ns** on the command line would affect only the *tpd* generic on the */top/u1* instance, assigning it a value of 20ns. Specifying **-Gu1/tpd=20ns** affects the *tpd* generic on all instances named *u1*. Specifying **-Gtpd=20ns** affects all generics named *tpd*.

<Value> — Specifies an appropriate value for the declared data type of a VHDL generic or any legal value for a Verilog parameter. Make sure the value you specify for a VHDL generic is appropriate for VHDL declared data types. Integers are treated as signed values. For example, **-Gp=-10** overwrites the parameter *p* with the signed value of -10.

- **-gblso <filename>**

On UNIX platforms, loads PLI/FLI shared objects with global symbol visibility. Essentially all data and functions are exported from the specified shared object and are available to be referenced and used by other shared objects. This option may also be specified with the [GlobalSharedObjectsList](#) variable in the *modelsim.ini* file. Optional.

- **-geometry <geometry_spec>**

Specifies the size and location of the main window. Optional. Where **<geometry_spec>** is of the form:

```
WxH+X+Y
```

- **-gui**
Starts the ModelSim GUI without loading a design and redirects the standard output (stdout) to the GUI Transcript window. Optional.
- **-help**
Displays the command's options and arguments. Optional.
- **-i**
Specifies that the simulator is to be run in interactive mode. Optional.
- **-installcolormap**
For UNIX only. Causes **vsim** to use its own colormap so as not to hog all the colors on the display. This is similar to the **-install** switch on Netscape. Optional.
- **-keeploaded**
Prevents the simulator from unloading/reloading any FLI/PLI/VPI shared libraries when it restarts or loads a new design. Optional. The shared libraries will remain loaded at their current positions. User application code in the shared libraries must reset its internal state during a restart in order for this to work effectively.
- **-keeploadedrestart**
Prevents the simulator from unloading/reloading any FLI/PLI/VPI shared libraries during a restart. Optional. The shared libraries will remain loaded at their current positions. User application code in the shared libraries must reset its internal state during a restart in order for this to work effectively.

We recommend using this option if you'll be doing warm restores after a restart and the user application code has set callbacks in the simulator. Otherwise, the callback function pointers might not be valid if the shared library is loaded into a new position.
- **-keepstdout**
For use with foreign programs. Instructs the simulator to not redirect the stdout stream to the Main window. Optional.
- **-l <filename>**
Saves the contents of the Transcript window to <filename>. Optional. Default is taken from the **TranscriptFile** variable (initially set to *transcript*) in the *modelsim.ini*. You can also specify "stdout" or "stderr" as <filename>.
- **-L <library_name> ...**
Specifies the library to search for design units instantiated from Verilog and for VHDL default component binding. Refer to "**Library Usage**" for more information. If multiple libraries are specified, each must be preceded by the **-L** option. Libraries are searched in the order in which they appear on the command line.

- `-learn <root_file_name>`

(Must be specified with `-novopt`) Specifies that you want the simulator to generate control files for retaining the proper level of visibility when performing an optimized simulation. The files generated are:

```
top_pli_learn.acc  
top_pli_learn.ocf  
top_pli_learn.ocm
```

Refer to “[Preserving Design Visibility with the Learn Flow](#)” in the User’s Manual for more information.

- `-Lf <library_name> ...`

Same as `-L` but libraries are searched before ‘`uselib`’ directives. Refer to “[Library Usage](#)” for more information. Optional.

- `-L mtiPA`

Use library of precompiled Power Aware behavioral models. Optional.

- `-lib <libname>`

Specifies the default working library where **vsim** will look for the design unit(s). Optional. Default is “work”.

- `<license_option>`

Restricts the search of the license manager. Optional. Use one of the license options listed below.

You can specify a license option only when invoking **vsim** from a UNIX/Linux shell command line, DOS command shell command line, or a Target for a Windows desktop shortcut. If you specify a license option from within the GUI, you will receive a message informing you of the error.

<license_option>	Description
<code>-lic_lnl_only</code>	check out msimhdlsim license only
<code>-lic_mixed_only</code>	check out msimhdlsim/msimhdlmix licenses only
<code>-lic_no_lnl</code>	exclude msimhdlsim license
<code>-lic_no_mix</code>	exclude msimhdlmix license
<code>-lic_no_slvhdl</code>	exclude qhsimvh license
<code>-lic_no_slvlog</code>	exclude qhsimvl license
<code>-lic_noqueue</code>	do not wait in queue when license is unavailable
<code>-lic_plus</code>	check out PLUS (VHDL and Verilog) license immediately after invocation
<code>-lic_vhdl</code>	check out VHDL license immediately after invocation
<code>-lic_vlog</code>	check out VLOG license immediately after invocation

You can also specify these options with the [License](#) variable in the *modelsim.ini* file. Note that settings made from the command line are additive to options set in the License variable.

For a complete list of license features and descriptions, see the *Installation & Licensing Guide*.

- `-load_elab <filename>`

Loads an elaboration file that was created with **-elab**. Optional. Refer to “[Simulating with an Elaboration File](#)” for more information.

You can not use this switch with any form of the `-memprof` switch. To analyze memory usage when simulating an elaboration file, you will need to use the [profile on](#) command. For example:

```
vsim -load_elab top.elab -do  
"profile on -m -fileonly top_mem_run.rpt; run -all"
```

- `-memprof`

Causes memory allocation data to be collected during elaboration and simulation. Shows what part of the design is using memory. Optional.

- `-memprof+call`

Unwinds the call stack and collects the call tree information. Optional. At the VSIM prompt, call stack collection can also be turned on with **profile option collect_calltrees on** and off with **profile option collect_calltrees off**.

- `-memprof+file=<filename>`

Saves memory profile data to the named file and makes the data available for viewing and reporting during the current simulation. The file can be used for archival or comparison purposes. Optional.

- `-memprof+fileonly=<filename>`

Saves memory profile data to the named file only. The file can be read in later with the [profile reload](#) command for analysis. This mode is useful for large designs, when the design plus internal profiling data would use up too much memory. Optional.

- `-msgmode both | tran | wlf`

Specifies the location(s) for the simulator to output elaboration and runtime messages. Refer to the section "[Message Viewer Window](#)" in the User's Manual for more information.

`both` — outputs messages to both the transcript and the WLF file. Default behavior

`tran` — outputs messages only to the transcript, therefore they are not available in the Message Viewer.

`wlf` — outputs messages only to the WLF file/Message Viewer, therefore they are not available in the transcript.

- **-modelsimini <ini_filepath>**

Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

- **-multisource_delay min | max | latest**

Controls the handling of multiple PORT or INTERCONNECT constructs that terminate at the same port. Optional. By default, the Module Input Port Delay (MIPD) is set to the max value encountered in the SDF file. Alternatively, you may choose the min or latest of the values. If you have a Verilog design and want to model multiple interconnect paths independently, use the **+multisource_int_delays** argument.

- **+multisource_int_delays**

Enables multisource interconnect delay with pulse handling and transport delay behavior. Works for both Verilog and VITAL cells. Optional.

Use this argument when you have interconnect data in your SDF file and you want the delay on each interconnect path modeled independently. Pulse handling is configured using the **+pulse_int_e** and **+pulse_int_r** switches (described below).

The **+multisource_int_delays** argument cannot be used if you compiled using the **-novital** argument to **vcom**. The **-novital** argument instructs **vcom** to implement VITAL functionality using VHDL code instead of accelerated code, and multisource interconnect delays cannot be implemented purely within VHDL.

- **-name <name>**

Specifies the application name used by the interpreter for send commands. This does not affect the title of the window. Optional.

- **-noassume**

Disables simulation of PSL and SystemVerilog assume directives. Optional. This option may also be specified with the [SimulateAssumeDirectives](#) variable in the *modelsim.ini* file. Refer to “[Processing Assume Directives](#)” for more information.

- **-no_autoacc**

Prevents **vsim** from automatically passing the **+acc** switch to **vopt**. Optional. By specifying this argument you can prevent **vopt** from opening any Verilog PLI modules for accessibility. You can pass specific **+acc** options to **vopt** by using the **-voptargs** argument.

- **-noautoldlibpath**

Disables the default internal setting of LD_LIBRARY_PATH, enabling you to set it yourself. Optional. Use this argument to make sure that LD_LIBRARY_PATH is not set automatically while you are using the GUI,

- **-nocapacity**
Disables the display of both coarse-grain and fine-grain analysis of memory capacity. Optional.
- **-nocompress**
Causes VSIM to create uncompressed checkpoint files. Optional. This option may also be specified with the [CheckpointCompressMode](#) variable in the *modelsim.ini* file.
- **-nocrossautobins**
Prevents the automatic generation of cross bins for covergroups. Cross bins are excluded from coverage computation and coverage reports. Optional.
- **-noimmedca**
Causes Verilog event ordering to occur without enforced prioritization—continuous assignments and primitives are not run before other normal priority processes scheduled in the same iteration. Use this argument to prevent the default event ordering where continuous assignments and primitives are run with “immediate priority.” Optional.
- **+no_notifier**
Disables the toggling of the notifier register argument of all timing check system tasks. Optional. By default, the notifier is toggled when there is a timing check violation, and the notifier usually causes a UDP to propagate an X. This argument suppresses X propagation in both Verilog and VITAL for the entire design.

You can suppress X propagation on individual instances using the [tcheck_set](#) command.
- **-nopsl**
Instructs ModelSim to ignore any PSL assertions that were compiled with the design. By default **vsim** automatically invokes the PSL assertion engine at runtime if any assertions were compiled with the design.
- **+nospecify**
Disables specify path delays and timing checks in Verilog. Optional.
- **-nosva**
Instructs ModelSim to ignore any SystemVerilog concurrent assertions that were compiled with the design. Optional. By default, **vsim** automatically invokes the assertion engine at runtime if any assertions were compiled with the design.
- **+no_tchk_msg**
Disables error messages generated when timing checks are violated. Optional. For Verilog, it disables messages issued by timing check system tasks. For VITAL, it overrides the `MsgOn` arguments and generics.

Notifier registers are still toggled and may result in the propagation of Xs for timing check violations.

You can disable individual messages using the [tcheck_set](#) command.

- `-note <msg_number>[,<msg_number>,...]`

Changes the severity level of the specified message(s) to "note." Optional. Edit the [note](#) variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.
- `+notimingchecks`

Disables Verilog timing checks. (This option sets the generic TimingChecksOn to FALSE for all VHDL Vital models with the Vital_level0 or Vital_level1 attribute. Generics with the name TimingChecksOn on non-VITAL models are unaffected.) Optional. By default, Verilog timing check system tasks (`$setup`, `$hold`,...) in specify blocks are enabled. For VITAL, the timing check default is controlled by the ASIC or FPGA vendor, but most default to enabled.

You can disable individual checks using the [tcheck_set](#) command.
- `-novopt`

Prevents ModelSim from running the [vopt](#) command automatically. If you have the [VoptFlow](#) variable set to 1 (optimizations turned on) in the *modelsim.ini* file, **vsim** automatically runs **vopt** if you didn't invoke it manually. If you specify this argument, you should be sure to specify it with your compilation command (`vcom` or `vlog`). One scenario in which you may want to use this switch is when coding an RTL block with a small testcase.
- `-OVMdebug`

Enables the creation of information for using the OVM-aware debugging windows in the GUI. Refer to the section “[OVM-Aware Debug](#)” in the User's Manual.
- `-pa`

Performs the verification in power aware mode. Specific to [Power Aware verification](#).
- `-pa_allowzerotimeevent`

Enables corruption, isolation, or release of signals at time 0.
- `-pa_bboxprefix= <pathname>`

Changes the hierarchical prefix to the DUT of a Power Aware analysis at simulation, where `<pathname>` is the full path to the Verilog or VHDL design element under test (including the instantiation name). Note that you use this argument with the `vopt -pa_bbox` command and argument for the first time on the specified DUT. You can then use this argument to change ModelSim Power Aware analyses at simulation runtime without having to re-run `vopt` on the same DUT. That is, you can run a simulation with a different top hierarchy to the DUT that was passed with `vopt` (by changing the root of DUT hierarchy to `<pathname>`) to create Power Aware results without running another `vopt` command.
- `-pa_gls <testbench_top>`

Enables gate-level simulation for a Power Aware analysis, performed on the top-of-design test bench (`testbench_top`).

- `-pa_lib <pathname>`
Specifies the library location where Power Aware information from the `vopt` command is stored. Use this argument when performing gate-level simulation.
- `-pa_zcorrupt`
Changes the default corruption value used at power-down from 'X' to 'Z'. This helps make debugging easier and also helps to differentiate an X value forced by Power Aware (now Z) from an X intentionally specified in the RTL design. At power-up, these elements would take either the X value or their RTL value, depending on their set, reset, clock, or retention specifications.

This change affects the following types:

VHDL — `std_logic`, `std_ulogic`, and their vectors

Verilog — `reg`, `wire`, and their 1-d arrays (Verilog memories (2-D array) are not supported)

- `-permissive`
Allows messages in the LRM group of error messages to be downgraded to a warning. Optional.
- `-plicompatdefault [latest | 2005 | 2001]`
Specifies the VPI object model behavior within vsim. This switch applies globally, not to individual libraries.
 - latest — This is equivalent to the "2009" argument. This is the default behavior if you do not specify this switch or if you specify the switch without an argument.
 - 2009 — Instructs vsim to use the object models as defined in IEEE Std P1800-2009 (unapproved draft standard). You can also use "09" as an alias.
 - 2005 — Instructs vsim to use the object models as defined in IEEE Std 1800-2005 and IEEE Std 1364-2005. You can also use "05" as an alias.
 - 2001 — Instructs vsim to use the object models as defined in IEEE Std 1364-2001. When you specify this argument, SystemVerilog objects will not be accessible. You can also use "01" as an alias.

You can also control this behavior with the [PliCompatDefault](#) variable in the `modelsim.ini` file, where the `-plicompatdefault` argument will override the `PliCompatDefault` variable.

You should note that there are a few cases where the 2005 VPI object model is incompatible with the 2001 model, which is inherent in the specifications.

Refer to the appendix "[Verilog Interfaces to C](#)" in the User's Manual for more information.

- `-printsimstats`
Prints the output of the [simstats](#) command to the screen at the end of simulation before exiting. Edit the [PrintSimStats](#) variable in the `modelsim.ini` file to set the simulation to print the `simstats` data by default.

- **-psl**
Instructs ModelSim to elaborate all PSL assertion directives. Optional. Overrides the [SimulatePSL](#) *modelsim.ini* file setting. See also **vsim -nopsl**.
- **-psloneattempt**
Force a single attempt to start at the beginning of simulation in order to test PSL directives with top level "always/never" properties. As per strict 1850-2005 PSL LRM, an always/never property can either pass or fail. However, by default, QuestaSim reports multiple passes and/or failures, which corresponds to multiple attempts made while executing a top level "always/never" property. With this switch, the directive will either match (pass), fail, or vacuously-match (provided it is not disabled/aborted). If the "always/never" property fails, the directive is immediately considered a failure and the simulation will not go further. If there is no failure (or disable/abort) until end of simulation then a match (pass) is reported. You can turn this feature on permanently using the [PslOneAttempt](#) variable in the *modelsim.ini* file.
- **-pslinfinitethreshold=<integer>**
Specifies the number of clock ticks that represent infinite clock ticks. Only affects PSL strong operators, namely: eventually!, until! and until_!. If the end of simulation is reached and an active strong-property has not clocked this number of clock ticks, neither pass nor fail (i.e. vacuous match) is returned; else, respective fail/pass is returned. The default value is '0' (zero) which effectively does not check for clock tick condition. You can turn on this feature using this switch or the [PslInfinityThreshold](#) in the *modelsim.ini* file.
- **+pulse_int_e/<percent>**
Controls how pulses are propagated through interconnect delays, where <percent> is a number between 0 and 100 that specifies the error limit as a percentage of the interconnect delay. Optional. Used in conjunction with **+multisource_int_delays** (see above). This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog.

A pulse greater than or equal to the error limit propagates to the output in transport mode (transport mode allows multiple pending transitions on an output). A pulse less than the error limit and greater than or equal to the rejection limit (see **+pulse_int_r/<percent>** below) propagates to the output as an X. If the rejection limit is not specified, then it defaults to the error limit. For example, consider an interconnect delay of 10 along with a **+pulse_int_e/80** option. The error limit is 80% of 10 and the rejection limit defaults to 80% of 10. This results in the propagation of pulses greater than or equal to 8, while all other pulses are filtered.
- **+pulse_int_r/<percent>**
Controls how pulses are propagated through interconnect delays, where <percent> is a number between 0 and 100 that specifies the rejection limit as a percentage of the interconnect delay. Optional. This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog.

A pulse less than the rejection limit is filtered. If the error limit is not specified by `+pulse_int_e` then it defaults to the rejection limit.

- `-quiet`
Disable 'Loading' messages during batch-mode simulation. Optional.
- `-restore <filename>`
Specifies that **vsim** is to restore a simulation saved with the [checkpoint](#) command. Optional.
You must restore vsim under the same environment in which you did the checkpoint. This means not only the same type of machine and OS and at least the same memory size, but also the same vsim environment such as GUI vs. command line mode.
- `-runinit`
Initializes non-trivial static SystemVerilog variables, for example expressions involving other variables and function calls, before displaying the simulation prompt.
- `+sdf_iopath_to_prim_ok`
Prevents **vsim** from issuing an error when it cannot locate specify path delays to annotate. If you specify this argument, IOPATH statements are annotated to the primitive driving the destination port if a corresponding specify path is not found. Optional. Refer to “[SDF to Verilog Construct Matching](#)” for additional information.
- `-sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=]<sdf_filename>`
(optional) Annotates VITAL or Verilog cells in the specified SDF file (a Standard Delay Format file) with minimum, typical, or maximum timing.
 - `@<delayScale>` — scales all values by the specified value. For example, if you specify `-sdfmax@1.5`, all maximum values in the SDF file are scaled to 150% of their original value.
Do not use this option if you scaled the SDF file while using the [sdfcom](#) command.
 - `<instance>=` — specifies a specific instance for the associated SDF file. Use this when not performing backannotation at the top level.
 - `<sdf_filename>` — specifies the file containing the SDF information.
- `-sdfminr | -sdftypr | -sdfmaxr[@<delayScale>] [<instance>=]<sdf_filename>`
(optional) Specifies when an instance of a black-boxed (vopt -bbox) module, which has an associated, default SDF file is to be re-annotated with minimum, typical, or maximum timing from the specified SDF file.
 - `@<delayScale>` — scales all values by the specified value. For example, if you specify `-sdfmax@1.5`, all maximum values in the SDF file are scaled to 150% of their original value.
Do not use this option if you scaled the SDF file while using the [sdfcom](#) command.
 - `<instance>=` — specifies a specific instance for the associated SDF file. Use this when not performing backannotation at the top level.
 - `<sdf_filename>` — specifies the file containing the SDF information.

Note



The simulator assumes that the instance/timing object hierarchy in the new SDF file is compatible with the SDF file specified during blackboxing with the `vopt` command.

The following is a simple usage flow:

```
# Assume module top contains three instances (u1, u2, and u3)
# of a black-boxed module bboxMod.
vlib work
vlog bboxMod.v

# blackbox bboxMod and annotate with sdf1.
vopt -bbox bboxMod -o bboxMod_opt -sdfmin bboxMod=sdf1

vlog top.v

# Use the default SDF file sdf1 for the blackbox instance of u1,
# but override the SDF for u2 and u3.
vsim top +sdf_verbose -sdftypr /top/u2=sdf2 -sdfmaxr /top/u3=sdf3
run -all
```

- `-sdfmaxerrors <n>`

Controls the number of Verilog SDF missing instance messages to be generated before terminating vsim. Optional. `<n>` is the maximum number of missing instance error messages to be emitted. The default number is 5.

- `-sdfnoerror`

Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue. Optional.

- `-sdfnowarn`

Disables warnings from the SDF reader. Optional. Refer to “[VHDL Simulation](#)” for an additional discussion of SDF.

- `+sdf_report_unannotated_insts`

Enables error messages for any un-annotated Verilog instances with specify blocks or VHDL instances with VITAL timing generics that are under regions of SDF annotation.

- `+sdf_verbose`

Turns on the verbose mode during SDF annotation. The Transcript window provides detailed warnings and summaries of the current annotation as well as information including the module name, source file name and line number. Optional.

- `-suppress <msg_number>[,<msg_number>,...]`

Prevents the specified message(s) from displaying. Optional. You cannot suppress Fatal or Internal messages. Edit the `suppress` variable in the `modelsim.ini` file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

- **-sva**
Instructs ModelSim to elaborate all SVA concurrent assertion directives. Optional. Overrides the **SimulateSVA** *modelsim.ini* file setting. See also **vsim -nosva**.
- **-sync**
Executes all X server commands synchronously, so that errors are reported immediately. Does not apply to Windows platforms.
- **-t [<multiplier>]<time_unit>**
Specifies the simulator time resolution. Optional. <time_unit> must be one of the following:

fs, ps, ns, us, ms, sec

The default is 1ns; the optional <multiplier> may be 1, 10 or 100. Note that there is no space between the multiplier and the unit (for example, 10fs, not 10 fs).

If you omit the **-t** argument, the default simulator time resolution depends on design type:

- In a VHDL design—the value specified for the Resolution variable in *modelsim.ini* is used.
- In a Verilog design with ‘timescale directives—the minimum specified time precision of all directives is used.
- In a Verilog design with no ‘timescale directives—the value specified for the **Resolution** variable in the *modelsim.ini* file is used.
- In a mixed design with VHDL on top—the value specified for the Resolution variable in the *modelsim.ini* file is used.
- In a mixed design with Verilog on top—
 - for Verilog modules not under a VHDL instance: the minimum value specified for their ‘timescale directives is used.
 - for Verilog modules under a VHDL instance: all their ‘timescale directives are ignored (the minimum value for ‘timescale directives in all modules not under a VHDL instance is used).

If there are no ‘timescale directives in the design, the value specified for the Resolution variable in *modelsim.ini* is used.



Tip: After you have started a simulation, you can view the current simulator resolution by using the **report** command as follows:

report simulator state

- **-tab <tabfile>**
Specifies the location of a Synopsys VCS “tab” file (.tab), which the simulator uses to automate the registration of PLI functions in the design.

<tabfile> — The location of a *.tab* file contains information about PLI functions. The tool expects the *.tab* file to be based on Synopsys VCS version 7.2 syntax. Because the format for this file is non-standard, changes to the format are outside of the control of Mentor Graphics.

By specifying the location of a *.tab* file, you do not need to use the `-no_autoacc` switch, which prevents `vopt` from opening PLI modules for accessibility.

If you are using the Two-step optimization flow, the tool passes this information automatically to `vopt`, which uses the file to improve accessibility rules.

If you are using the Three-step optimization flow, you must specify this switch on both the `vopt` and `vsim` command lines.

You can use this switch when you disable optimization with the `-novopt` switch.

- `-tag <string>`

Specifies a string tag to append to foreign trace filenames. Optional. Used with the **`-trace_foreign <int>`** option. Used when running multiple traces in the same directory.

See the *ModelSim FLI Reference* for more information.

- `-title <title>`

Specifies the title to appear for the ModelSim Main window. Optional. If omitted the current ModelSim version is the window title. Useful when running multiple simultaneous simulations. Text strings with spaces must be in quotes (e.g., "my title").

- `-togglecountlimit <int>`

Specifies the global toggle coverage count limit for toggle nodes in an entire simulation. Optional. Overrides the global value set by the `ToggleCountLimit` *modelsim.ini* variable. If used, it provides default limit values for any design units not compiled with either `vlog -togglecountlimit` or `vcom -togglecountlimit`. If any design units were compiled with those arguments, those values apply during simulation unless you use the `toggle add -countlimit` command to override the values. After the limit is reached, further activity on the node is ignored for toggle coverage. All possible transition edges must reach this count for the limit to take effect. For example, if you are collecting toggle data on 0->1 and 1->0 transitions, both transition counts must reach the limit. If you are collecting "full" data on 6 edge transitions, all 6 must reach the limit.

- `-togglewidthlimit <int>`

Sets the maximum width of signals, <int>, that are automatically added to toggle coverage with the `-cover t` argument for `vcom` or `vlog`. Optional. Overrides the global value set by the `ToggleWidthLimit` *modelsim.ini* variable. If used, it provides default limit values for any design units not compiled with `vlog -togglewidthlimit` or `vcom -togglewidthlimit`.

- `-trace_foreign <int>`

Creates two kinds of foreign interface traces: a log of what functions were called, with the value of the arguments, and the results returned; and a set of C-language files to replay what the foreign interface side did.

The purpose of the logfile is to aid the debugging of your FLI/PLI/VPI code. The primary purpose of the replay facility is to send the replay file to MTI support for debugging co-simulation problems, or debugging problems for which it is impractical to send the FLI/PLI/VPI code. See the *ModelSim FLI Reference* for more information.

- +transport_int_delays

Selects transport mode with pulse control for single-source nets (one interconnect path). Optional. By default interconnect delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through interconnect delays.

This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog. This option works independently from +multisource_int_delays.

- -vcdstim [<instance>=]<filename>

Specifies a VCD file from which to re-simulate the design. Optional. The VCD file must have been created in a previous ModelSim simulation using the [vcd dumpports](#) command. Refer to “[Using Extended VCD as Stimulus](#)” for more information.

- -version

Returns the version of the simulator as used by the licensing tools. Optional.

- -view [<alias_name>=]<WLF_filename>

Specifies a wave log format (WLF) file for vsim to read. Allows you to use vsim to view the results from an earlier simulation. The Structure, Objects, Wave, and List windows can be opened to look at the results stored in the WLF file (other ModelSim windows will not show any information when you are viewing a dataset).

<alias_name> — Specifies an alias for <WLF_file_name> where the default is to use the prefix of the WLF_filename. Wildcard characters are allowed.

<WLF_file_name> — Specifies the pathname of a saved WLF file.

See additional discussion in the Examples.

- -viewcov [<dataset_name>=]<UCDB_filename>

Invokes vsim in the coverage view mode to display UCDB data.

- -visual <visual>

Specifies the visual to use for the window. Optional. Does not apply to Windows platforms.

Where <visual> may be:

<class> <depth> — One of the following:

{directcolor | grayscale | greyscale | pseudocolor | staticcolor | staticgray | staticgrey | truecolor}

followed by:

<depth> — Specifies how many bits per pixel are needed for the visual.

default — Instructs the tool to use the default visual for the screen

<number> — Specifies a visual X identifier.

best <depth> — Instructs the tool to choose the best possible visual for the specified <depth>, where:

<depth> — Specifies how many bits per pixel are needed for the visual.

- +vlog_retain_on | +vlog_retain_off

Enables or disables SDF RETAIN delay processing. +vlog_retain_on is the default behavior. Refer to the section “[Retain Delay Behavior](#)” in the User’s Manual for more information.

- +vlog_retain_same2same_on | +vlog_retain_same2same_off

Enables or disables SDF RETAIN delay processing of X insertion on outputs that do not change, but the causal inputs change. +vlog_retain_same2same_on is the default behavior. Refer to the section “[Retain Delay Behavior](#)” in the User’s Manual for more information.

- -vopt

Instructs **vsim** to run the **vopt** command automatically if vopt was not manually invoked. Not needed unless the **VoptFlow** variable has been set to 0 in the *modelsim.ini*. Optional. Refer to the chapter entitled “[Optimizing Designs with vopt](#)” for more information.

- -voptargs="<args>"

Specifies arguments that **vsim** should pass to **vopt** when running vopt automatically. Any arguments must be separated by spaces, just as you would on the vopt command line.

The primary purpose of this argument is to pass +acc arguments. Optional.

- -vopt_verbose

Displays **vopt** messages in the Transcript window. Optional. By default, these messages are not displayed or saved when **vopt** is run using **vsim**.

- -warning <msg_number>[,<msg_number>,...]

Changes the severity level of the specified message(s) to "warning." Optional. Edit the **warning** variable in the *modelsim.ini* file to set a permanent default. Refer to “[Changing Message Severity Level](#)” for more information.

- -wlf <file_name>

Specifies the name of the wave log format (WLF) file to create. The default is *vsim.wlf*. Optional. This option may also be specified with the **WLFfilename** variable in the *modelsim.ini* file.

- -wlfcache size <n>

Specifies the size in megabytes of the WLF reader cache. Optional. By default the cache size is set to zero. WLF reader caching caches blocks of the WLF file to reduce redundant file I/O. This should have significant benefit in slow network environments. This option may also be specified with the **WLFCacheSize** variable in the *modelsim.ini* file.

- **-wlfcollapsedelta**
Instructs ModelSim to record values in the WLF file only at the end of each simulator delta step. Any sub-delta values are ignored. May dramatically reduce WLF file size. This option may also be specified with the [WLFCollapseMode](#) variable in the *modelsim.ini* file. Default.
- **-wlfcollapsetime**
Instructs ModelSim to record values in the WLF file only at the end of each simulator time step. Any delta or sub-delta values are ignored. May dramatically reduce WLF file size. This option may also be specified with the [WLFCollapseMode](#) variable in the *modelsim.ini* file. Optional.
- **-wlfcompress**
Creates compressed WLF files. Default. Use **-wlfnocompress** to turn off compression. This option may also be specified with the [WLFCompress](#) variable in the *modelsim.ini* file.
- **-wlfdeleteonquit**
Deletes the current simulation WLF file (*vsim.wlf*) automatically when the simulator exits. Optional. This option may also be specified with the [WLFDeleteOnQuit](#) variable in the *modelsim.ini* file.
- **-wlfindex**
(optional) Enables indexing of the WLF file. Indexing enables fast seeking to specific times. Makes viewing of wave data faster, however performance during optimization will be slower because indexing and optimization require significant memory and CPU resources. Indexing may also be specified with the [WLFIndex](#) variable in the *modelsim.ini* file.
(default)
- **-wlfnoindex**
Disables indexing and optimization of the WLF file. Disabling indexing makes viewing wave data slow unless the display is near the start of the WLF file. May provide a significant performance boost when archiving WLF files. Indexing and optimization information can be added back to the file using [wlfman](#) optimize.
- **-wlflock**
Locks a WLF file. Optional. An invocation of ModelSim will not overwrite a WLF file that is being written by a different invocation.
- **-wlfnocollapse**
Instructs ModelSim to preserve all events for each logged signal and their event order to the WLF file. May result in relatively larger WLF files. This option may also be specified with the [WLFCollapseMode](#) variable in the *modelsim.ini* file. Optional.
- **-wlfnocompress**
Causes **vsim** to create uncompressed WLF files. Optional. WLF files are compressed by default in order to reduce file size. This may slow simulation speed by one to two percent. You may want to disable compression to speed up simulation or if you are experiencing

problems with faulty data in the resulting WLF file. This option may also be specified with the [WLFCompress](#) variable in the *modelsim.ini* file.

- **-wlfnodeleteonquit**

Preserves the current simulation WLF file (*vsim.wlf*) when the simulator exits. Default. This option may also be specified with the [WLFDeleteOnQuit](#) variable in the *modelsim.ini* file.

- **-wlfnolock**

Disables WLF file locking. Optional. This will prevent vsim from checking whether a WLF file is locked prior to opening it as well as preventing vsim from attempting to lock a WLF once it has been opened.

- **-wlfnoopt**

Disables optimization of waveform display in the Wave window. Optional. This option may also be specified with the [WLFOptimize](#) variable in the *modelsim.ini* file.

- **-wlfopt**

Optimizes the WLF file. Enables faster display of waveforms in the Wave window when the display is zoomed out to display a larger time range. Default. Optional. This option may also be specified with the [WLFOptimize](#) variable in the *modelsim.ini* file.

- **-wlfsimcachesize <n>**

Specifies the size in megabytes of the WLF reader cache for the current simulation dataset only. Optional. By default the cache size is set to zero. This makes it easier to set different sizes for the WLF reader cache used during simulation and those used during post-simulation debug. WLF reader caching caches blocks of the WLF file to reduce redundant file I/O. If neither **-wlfsimcachesize** nor [WLFSimCacheSize](#) *modelsim.ini* variable are specified, the **-wlfcachesize** or [WLFCacheSize](#) settings will be used.

- **-wflslim <size>**

(optional) Specifies a size restriction for the event portion of the WLF file.

size — an integer, in megabytes, where the default is 0, which implies an unlimited size.

Note that a WLF file contains event, header, and symbol portions. The size restriction is placed on the event portion only. When ModelSim exits, the entire header and symbol portion of the WLF file is written. Consequently, the resulting file will be larger than the specified size.

QuestaSim uses 64-bit file I/O for maintaining the WLF file, which allows access to file systems supporting up to 2^{63} -byte files. File size limitations are also governed by the OS file system in use as well as per-process limits on file size. You can determine any per-process limit by using the following shell commands:

- sh/bash/ksh: `ulimit -a`
- csh/tcsh: `limit`

If used in conjunction with **-wlftlim**, the more restrictive of the limits takes precedence.

This option may also be specified with the [WLFSizeLimit](#) variable in the *modelsim.ini* file. (See [Limiting the WLF File Size](#).)

- `-wlfthreads | -wlfnothreads`

Specifies whether the logging of information to the WLF file is performed using multithreading.

This behavior is on (`-wlfthreads`) by default on Solaris and Linux platforms where there are more than one processor on the system. If there is only one processor available, or you are running on a Windows system, this behavior is off by default (`-wlfnothreads`).

When this behavior is enabled, the logging of information is performed on the secondary processor while the simulation and other tasks are performed on the primary processor.

You can turn this option off with the `-wlfnothreads` option, which you may want to do if you are performing several simulations with logging at the same time.

You can also control this behavior with the [WLFUseThreads](#) variable in the *modelsim.ini* file.

- `-wlftlim <duration>`

Specifies the duration of simulation time for WLF file recording. Optional. The default is infinite time (0). The `<duration>` is an integer of simulation time at the current resolution; you can optionally specify the resolution if you place curly braces around the specification. For example,

```
{5000 ns}
```

sets the duration at 5000 nanoseconds regardless of the current simulator resolution.

The time range begins at the current simulation time and moves back in simulation time for the specified duration. For example,

```
vsim -wlftlim 5000
```

writes at most the last 5000ns of the current simulation to the WLF file (the current simulation resolution in this case is ns).

If used in conjunction with **-wflslim**, the more restrictive of the limits will take effect.

This option may also be specified with the [WLFTimeLimit](#) variable in the *modelsim.ini* file.

The **-wflslim** and **-wlftlim** switches were designed to help users limit WLF file sizes for long or heavily logged simulations. When small values are used for these switches, the values may be overridden by the internal granularity limits of the WLF file format. (See [Limiting the WLF File Size](#).)

Arguments, VHDL

- `-absentisempty`

Causes VHDL files opened for read that target non-existent files to be treated as empty, rather than ModelSim issuing fatal error messages. Optional.

- **-foreign <attribute>**
Specifies the foreign module to load. Optional. <attribute> is a quoted string consisting of the name of a C function and a path to a shared library. For example,

```
vsim -foreign "c_init for.sl"
```

You can load up to ten foreign modules. Syntax for the attribute is further described in the Introduction chapter of the *ModelSim FLI Reference*.
- **-nocollapse**
Disables the optimization of internal port map connections. Optional.
- **-nofileshare**
Turns off file descriptor sharing. Optional. By default ModelSim shares a file descriptor for all VHDL files opened for write or append that have identical names.
- **-notoggleints**
Excludes VHDL integer values from toggle coverage. Overrides the [ToggleNoIntegers](#) modelsim.ini variable default behavior of on(1). Optional.
- **-noglitch**
Disables VITAL glitch generation. Optional.
Refer to “[VHDL Simulation](#)” for additional discussion of VITAL.
- **+no_glitch_msg**
Disable VITAL glitch error messages. Optional.
- **-std_input <filename>**
Specifies the file to use for the VHDL TextIO STD_INPUT file. Optional.
- **-std_output <filename>**
Specifies the file to use for the VHDL TextIO STD_OUTPUT file. Optional.
- **-strictvital**
Specifies to exactly match the VITAL package ordering for messages and delta cycles. Optional. Useful for eliminating delta cycle differences caused by optimizations not addressed in the VITAL LRM. Using this argument negatively impacts simulator performance.
- **-togglemaxintvalues <int>**
Specifies the maximum number of VHDL integer values to record for toggle coverage. Optional. This limit variable may be changed on a per-signal basis. The default value of <int> is 100 values.
- **-vcdread <filename>**
Simulates the VHDL top-level design from the specified VCD file. Optional. This argument is included for backwards compatibility. Consider using the **-vcdstim** argument instead. Refer to “[Simulating with Input Values from a VCD File](#)” for more details.

- **-vital2.2b**
Selects SDF mapping for VITAL 2.2b (default is VITAL 2000). Optional.

Arguments, Verilog

- **+alt_path_delays**
Configures path delays to operate in inertial mode by default. Optional. In inertial mode, a pending output transition is cancelled when a new output transition is scheduled. The result is that an output may have no more than one pending transition at a time, and that pulses narrower than the delay are filtered. The delay is selected based on the transition from the cancelled pending value of the net to the new pending value. The **+alt_path_delays** option modifies the inertial mode such that a delay is based on a transition from the current output value rather than the cancelled pending value of the net. This option has no effect in transport mode (see **+pulse_e/<percent>** and **+pulse_r/<percent>**).
- **-cvg63**
Forces compatibility with ModelSim 6.3 for coverage type-based coverage. Optional. Uses the *merge instances* mode. This option overrides the [SVCoverage63Compatibility](#) variable in the *modelsim.ini* file.
- **-cvghaltillbin**
Causes the simulation to halt whenever an illegal coverpoint or cross bin is hit during sampling. Optional.
- **-cvgmaxrpthscross**
Specifies maximum limit for the number of cross bin products in a coverage report and UCDB. Optional. This overrides the setting of the [MaxReportRhsSVCrossProducts](#) variable in the *modelsim.ini* file. The default value is 0.
- **-cvgsingledefaultbin**
Collapses a default array bin to a scalar bin. Optional. By default, a default array bin is modeled as a sparse array and the sub-bins are reported only when there are one or more samples.
- **+delayed_timing_checks**
Causes timing checks to be performed on the delayed versions of input ports (used when there are negative timing check limits). Optional. ModelSim automatically detects and applies **+delayed_timing_checks** to optimized cells with negative timing checks. To turn off this feature, specify **+no_autodtc** with vsim.
- **-dpiexportobj <objfile>**
Generates the C export wrappers and associated compiled object code for your design. The C wrapper code is written to your *<work>/_dpi/* directory, so it must have the proper permissions. The object file(s) are written to whatever location you specify with the *<objfile>* argument.

For Windows platforms, this is a required switch when using DPI that generates an *.obj* file suitable for linking into a *.dll*. Refer to “[DPI Use Flow](#)” for additional information.

For Linux and UNIX platforms, this switch generates both a *.o* and a *.so* file. The *.o* file is suitable for linking into a larger *.so* file, which may contain import code. The *.so* file can be used directly, for example as an argument to the **vsim -gblso** switch or as a dependent library in the link command for an import shared object.

Once you compile the export wrapper code into a shared object or *.dll*, you can manually load it into the simulation using **-sv_lib**, or perhaps **-gblso**. When you do manually load the export wrapper code, you should use the **-nodpiexports** switch so that the simulation does not automatically generate and load the *<work>/_dpi/exportwrapper.so* file, which would cause symbol collisions.

- **-hazards**

Enables event order hazard checking in Verilog modules (Verilog only). Optional. You must also specify this argument when you compile your design with **vlog**. Refer to “[Hazard Detection](#)” for more details.

Note



Using **-hazards** implicitly enables the **-compat** argument. As a result, using this argument may affect your simulation results.

- **+initmem+<seed>**

Specifies the seed value to be used by random initialization for Verilog designs. Random initialization (of only 0 or 1) occurs at runtime for memories compiled by **vlog** with the **+initmem** option without specifying a modifier (**+{0 | 1 | X | Z}**).

If no **+initmem** is present on the **vsim** command line, a random seed of 0 is used during initialization.

+<seed> — any signed 32-bit integer (-2147483648 to +2147483647).

- **+initreg+<seed>**

Specifies the seed value to be used by random initialization for Verilog designs. Random initialization (of only 0 or 1) occurs at runtime for registers compiled by **vlog** with the **+initreg** option without specifying a modifier (**+{0 | 1 | X | Z}**).

If no **+initreg** is present on the **vsim** command line, a random seed of 0 is used during initialization.

+<seed> — any signed 32-bit integer (-2147483648 to +2147483647).

- **+maxdelays**

Selects the maximum value in **min:typ:max** expressions. Optional. The default is the typical value. Has no effect if you specified the **min:typ:max** selection at compile time.

- **+mindelays**
Selects the minimum value in min:typ:max expressions. Optional. The default is the typical value. Has no effect if you specified the min:typ:max selection at compile time.
- **+no_autdte**
Turns off auto-detection of optimized cells with negative timing checks and auto-application of **+delayed_timing_checks** to those cells. Optional.
- **+no_cancelled_e_msg**
Disables negative pulse warning messages. Optional. By default **vsim** issues a warning and then filters negative pulses on specify path delays. You can drive an X for a negative pulse using **+show_cancelled_e**.
- **-nocvg63**
Abandons compatibility with ModelSim 6.3 for covergroup type-based coverage. Optional. Uses the *average of instances* mode, which is the default in IEEE Std 1800-2009 for SystemVerilog. This option overrides the **SVCovergroup63Compatibility** variable in the *modelsim.ini* file.
- **+no_neg_tchk**
Disables negative timing check limits by setting them to zero. Optional. By default negative timing check limits are enabled. This is just the opposite of Verilog-XL, where negative timing check limits are disabled by default, and they are enabled with the **+neg_tchk** option.
- **+no_notifier**
Disables the toggling of the notifier register argument of all timing check system tasks. Optional. By default, the notifier is toggled when there is a timing check violation, and the notifier usually causes a UDP to propagate an X. This argument suppresses X propagation on timing violations for the entire design.

You can suppress X propagation on individual instances using the **tcheck_set** command.
- **+no_path_edge**
Causes ModelSim to ignore the input edge specified in a path delay. Optional. The result of this argument is that all edges on the input are considered when selecting the output delay. Verilog-XL always ignores the input edges on path delays.
- **+no_pulse_msg**
Disables the warning message for specify path pulse errors. Optional. A path pulse error occurs when a pulse propagated through a path delay falls between the pulse rejection limit and pulse error limit set with the **+pulse_r** and **+pulse_e** options. A path pulse error results in a warning message, and the pulse is propagated as an X. The **+no_pulse_msg** option disables the warning message, but the X is still propagated.
- **-no_risefall_delaynets**
Disables the rise/fall delay net delay negative timing check algorithm. Optional. This argument is provided to return ModelSim to its pre-6.0 behavior where violation regions

must overlap in order to find a delay net solution. In 6.0 versions and later, ModelSim uses separate rise/fall delays, so violation regions need not overlap for a delay solution to be found.

- `+no_show_cancelled_e`

Filters negative pulses on specify path delays so they don't show on the output. Default. Use `+show_cancelled_e` to drive a pulse error state.

- `+no_tchk_msg`

Disables error messages issued by timing check system tasks when timing check violations occur. Optional. Notifier registers are still toggled and may result in the propagation of Xs for timing check violations.

You can disable individual messages using the [tcheck_set](#) command.

- `-nodpiexports [-dpiexportcheckref]`

(optional) Instructs the command to not generate C wrapper code for DPI export task and function routines found at elaboration time. More specifically, the command does not generate the *exportwrapper.so* shared object file in `<work>/_dpi/`.

`-dpiexportcheckref` — (optional) Issues a fatal error if the DPI export task and function object file is out of date.

```
Fatal: (vsim-3830) The DPI-C export task and function object file
generated by vsim -dpiexportobj is out of date. Regenerate the file
and then rerun simulation.
```

For a description on when you should use this switch, refer to the section “[Integrating Export Wrappers into an Import Shared Object](#)” in the User’s Manual.

- `-noexcludehiz`

Instructs ModelSim to include truth table rows that contain Hi-Z states in the coverage count. Without this argument, these rows are automatically excluded. Optional.

- `+nosdferror`

Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue. Optional.

- `+nosdfwarn`

Disables warnings from the SDF annotator. Optional.

- `+nospecify`

Disables specify path delays and timing checks. Optional.

- `+nowarnBSOB`

Disables run-time warning messages for bit-selects in initial blocks that are out of bounds.

- `+nowarn<CODE>`

Disables warning messages in the category specified by `<CODE>`. Optional. Warnings that can be disabled include the `<CODE>` name in square brackets in the warning message. For example:

```
** Warning: (vsim-3017) test.v(2): [TFMPC] - Too few port
connections. Expected <m>, found <n>.
```

This warning message can be disabled with `+nowarnTFMPC`.

- `+ntc_warn`

Enables warning messages from the negative timing constraint algorithm. Optional. By default, these warnings are disabled.

This algorithm attempts to find a set of delays for the timing check delayed net arguments such that all negative limits can be converted to non-negative limits with respect to the delayed nets. If there is no solution for this set of limits, then the algorithm sets one of the negative limits to zero and recalculates the delays. This process is repeated until a solution is found. A warning message is issued for each negative limit set to zero.

- `-oldvhdlforgennames`

Enables the use of a previous style of naming in VHDL for ... generate statement iteration names in the design hierarchy. The previous style is controlled by the value of the [GenerateFormat](#) value. The default behavior is to use the current style names, which is described in “[Naming Behavior of VHDL For Generate Blocks](#)” This argument duplicates the function of the [OldVhdlForGenNames](#) variable in `modelsim.ini` and will override the setting of that variable if it specifies the current style.

- `-onfinish ask | stop | exit`

Customizes the simulator shutdown behavior when it encounters `$finish` or `sc_stop()` in the design:

- **ask** —
 - In batch mode, the simulation exits.
 - In GUI mode, a dialog box pops up and asks for user confirmation on whether to quit the simulation.
- **stop** — stops simulation and leave the simulation kernel running
- **exit** — exits out of the simulation without a prompt

By default, the simulator exits in batch mode; prompts you in GUI mode. Edit the [OnFinish](#) variable in the `modelsim.ini` file to set the default operation of `$finish`.

- `-pli "<object list>"`

Loads a space-separated list of PLI shared objects. Optional. The list must be quoted if it contains more than one object. This is an alternative to specifying PLI objects in the

Veriuser entry in the *modelsim.ini* file, refer to [modelsim.ini Variables](#). You can use environment variables as part of the path.

- +<plusarg>

Arguments preceded with "+" are accessible by the Verilog PLI routine **mc_scan_plusargs()**. Optional.

- +pulse_e/<percent>

Controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the error limit as a percentage of the path delay. Optional.

A pulse greater than or equal to the error limit propagates to the output in transport mode (transport mode allows multiple pending transitions on an output). A pulse less than the error limit and greater than or equal to the rejection limit (see **+pulse_r/<percent>**) propagates to the output as an X. If the rejection limit is not specified, then it defaults to the error limit. For example, consider a path delay of 10 along with a **+pulse_e/80** option. The error limit is 80% of 10 and the rejection limit defaults to 80% of 10. This results in the propagation of pulses greater than or equal to 8, while all other pulses are filtered. Note that you can force specify path delays to operate in transport mode by using the **+pulse_e/0** option.

- +pulse_e_style_ondetect

Selects the "on detect" style of propagating pulse errors (see **+pulse_e**). Optional. A pulse error propagates to the output as an X, and the "on detect" style is to schedule the X immediately, as soon as it has been detected that a pulse error has occurred. "on event" style is the default for propagating pulse errors (see **+pulse_e_style_onevent**).

- +pulse_e_style_onevent

Selects the "on event" style of propagating pulse errors (see **+pulse_e**). Default. A pulse error propagates to the output as an X, and the "on event" style is to schedule the X to occur at the same time and for the same duration that the pulse would have occurred if it had propagated through normally.

- +pulse_r/<percent>

Controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the rejection limit as a percentage of the path delay. Optional.

A pulse less than the rejection limit is suppressed from propagating to the output. If the error limit is not specified by **+pulse_e** then it defaults to the rejection limit.

- +sdf_nocheck_celltype

Disables the error check a for mismatch between the CELLTYPE name in the SDF file and the module or primitive name for the CELL instance. It is an error if the names do not match. Optional.

- `+show_cancelled_e`

Drives a pulse error state ('X') for the duration of a negative pulse on a specify path delay. Optional. By default ModelSim filters negative pulses.
- `-solveengine {auto | bdd | act}`

Selects a solver “engine” to be used for constrained random generation. Optional. Overrides the setting for [SolveEngine](#) in the *modelsim.ini* file.

 - `auto` — (default) ModelSim automatically selects the best engine for the current randomization operation.
 - `bdd` — evaluates all randomization operations using the Binary Decision Diagram (BDD) solver engine. The BDD-based solver is usually more efficient with sets of constraints involving certain bitwise logical relationships.
 - `act` — evaluates all randomization operations using the Arithmetic Constraint Tree (ACT) solver engine. The ACT-based solver can exhibit superior performance with constraints that entail heavy use of arithmetic operators (+, -, *, /).
- `-solvefaildebug`

Enable feature to debug SystemVerilog `randomize()` failures. Optional. Whenever a `randomize()` failure is detected during simulation, ModelSim will display the minimum set of constraints that caused the `randomize()` call to fail.

The simulator also prints to the transcript a simplified testcase of the failing randomization scenario. You can use this testcase to reproduce the failure or as an aid in debugging the constraints.
- `-solveflags=<flags>`

Modifies the behavior of the constraint solver (SystemVerilog `randomize`) to improve the evaluation performance of some types of constraints. Optional. You can change the default behavior of the constraint solver by setting the [SolveFlags](#) variable in the *modelsim.ini* file.

Valid `<flags>` are:

 - `i` — disable bit interleaving for `>`, `>=`, `<`, `<=` constraints
 - `n` — disable bit interleaving for all constraints
 - `r` — reverse bit interleaving
- `-solverev <version>`

Specifies random sequence generation compatibility with a prior letter release for the SystemVerilog solver. (It does not apply to the SystemC/SCV solver.) This argument is used to get the same random sequences during simulation as a prior letter release. The `<version>` is a string of a release number and letter, such as 6.2a. (`-solverev 6.2a`). Only prior letter releases (within same number release) are allowed. For example, in 6.2b you can specify `-solverev 6.2` or `-solverev 6.2a`, but cannot specify `-solverev 6.1f`. See also, the *modelsim.ini* ["SolveRev"](#) variable.

- `-sv_lib <shared_obj>`
Specifies the name of the DPI shared object with no extension. Required for use with DPI import libraries. Refer to “[DPI Use Flow](#)” for additional information.
- `-sv_liblist <filename>`
Specifies the name of a bootstrap file containing names of DPI shared objects to load. Optional.
- `-sv_root <dirname>`
Specifies the directory name to be used as the prefix for DPI shared object lookups. Optional.
- `-sv_seed <integer> | random`
Seeds the root random number generator for SystemVerilog threads with either a user-specified integer or a random number generated by ModelSim. Also seeds the `shuffle()` array ordering operator. Optional. The integer may be any signed 32-bit integer (-2147483648 to +2147483647). Integers outside the value range will be assigned either the minimum or maximum value. Edit the `Sv_Seed` variable in the `modelsim.ini` file to set a permanent default. Refer to “[Seeding the Random Number Generator \(RNG\)](#)” for more information.
- `-togglefixedsizearray | -notogglefixedsizearray`
The `-togglefixedsizearray` argument includes SystemVerilog unpacked fixed-size arrays, as well as VHDL multi-dimensional arrays and arrays-of-arrays, in toggle coverage. By default, packed fixed-size arrays are excluded. The `-togglefixedsizearray` argument overrides the `ToggleFixedSizeArray` `modelsim.ini` variable default setting of off (0). Optional.
- `-togglemaxfixedsizearray <int>`
Specifies the maximum size for the SystemVerilog unpacked real type fixed-size arrays, as well as VHDL multi-dimensional arrays and arrays-of-arrays, that are collected for toggle coverage. By default, large fixed-sized arrays (>1024 elements) are not included in toggle coverage, even when the `-togglefixedsizearray` option is used, as this can have an adverse impact on simulation performance. Use the `ToggleMaxFixedSizeArray` `modelsim.ini` variable to control this limit. Optional.
- `-togglemaxrealvalues <int>`
Specifies the maximum number of SystemVerilog real values to record for toggle coverage of a given signal. Optional. This limit variable may be changed on a per-signal basis. The default value of 100 values can be modified by editing the `ToggleMaxRealValues` `modelsim.ini` variable.
- `-togglepackedasvec`
Specifies that SystemVerilog packed structures and multi-d arrays are treated as flattened vectors for toggle coverage. Overrides the `TogglePackedAsVec` `modelsim.ini` variable default setting of off (0). Optional.

- **-togglevlogenumbits**
Specifies that SystemVerilog enum types are treated as reg-vectors for toggle coverage. Overrides the default setting of the [ToggleVlogEnumBits variable in modelsim.ini](#), which is off(0). Optional.
- **-togglevlogints | -notogglevlogints**
By default, SystemVerilog integer types (shortint, int, longint, byte, integer and time) are treated as reg-vectors, and counts are kept for each bit. The -notogglevlogints argument excludes these type from coverage, overriding the default setting of the [ToggleVlogIntegers variable in modelsim.ini](#), which is on(1). The -togglevlogints argument is used to enable coverage after it has been disabled. Optional.
- **-togglevlogreal | -notogglevlogreal**
The -togglevlogreal argument includes Verilog real value types in toggle coverage. Overrides the default setting of the [ToggleVlogReal variable in modelsim.ini](#), which is off (0). Optional.
- **+transport_path_delays**
Selects transport mode for path delays. Optional. By default, path delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through path delays. Note that this option affects path delays only, and not primitives. Primitives always operate in inertial delay mode.
- **+typdelays**
Selects the typical value in min:typ:max expressions. Default. Has no effect if you specified the min:typ:max selection at compile time.
- **-unattempteddimmed**
(optional) Causes any unexecuted immediate assertions to be considered in the coverage calculations for Total Coverage that are displayed in the GUI or in the coverage report. By default, any unexecuted immediate assertions are not included in the displayed calculations. This functionality can be set for all simulations through the [UnattemptedImmediateAssertions modelsim.ini](#) file variable.
- **-v2k_int_delays**
Causes interconnect delays to be visible at the load module port per the IEEE 1364-2001 spec. Optional. By default ModelSim annotates INTERCONNECT delays in a manner compatible with Verilog-XL. If you have \$sdf_annotate() calls in your design that are not getting executed, add the Verilog task \$sdf_done() after your last \$sdf_annotate() to remove any zero-delay MIPDs that may have been created. May be used in tandem with the **+multisource_int_delays** argument (see above).
Refer to [sdfcom](#) for SDF compilation information.

Arguments, SystemC

- **-scdpidebug**

Enables DPI debug single-stepping across SystemC-SystemVerilog call boundaries for SystemVerilog breakpoints placed inside an export function call that was initiated from an SC_METHOD.

Refer to the sections "[Setting Breakpoints](#)" and "[Stepping in C Debug](#)" for more information.

Turns on debugging support for DPI out-of-the-blue calls from a SystemC method when combined with the **vsim** argument **-dpioutoftheblue**. Refer to **-dpioutoftheblue** for more information.

- **-sclib** <library>

Specifies the design library where the SystemC shared library is created. By default, the SystemC shared library is created in the logical work library. This option is only necessary when the shared library is compiled in a design library other than the logical work directory (via **sccom -link -work** <lib>). For more information on the **sccom -link** and **-work** arguments, see [sccom](#).

- **-sc_arg** <string> ...

Specifies a string representing a startup argument which is subsequently accessible from within SystemC via the `sc_argc()` and `sc_argv()` functions (refer to "[Accessing Command-Line Arguments](#)").

If multiple SystemC startup arguments are specified, each must have a separate **-sc_arg** argument. SystemC startup arguments returned via `sc_argv()` are in the order in which they appear on the command line. White space within the <string> will not be treated specially, and the string, white space and all, will be accessible as a single string among the strings returned by `sc_argv()`.

Arguments, object

The object arguments may be a [<library_name>].<design_unit>, an *.mpf* file, a *.wlf* file, or a text file. Multiple design units may be specified for Verilog modules and mixed VHDL/Verilog configurations.

- <library_name>.<design_unit>

Specifies a library and associated design unit; multiple library/design unit specifications can be made. Optional. If no library is specified, the **work** library is used. You cannot use the wildcard * for this argument. Environment variables can be used. <design_unit> may be one of the following:

<configuration>	Specifies the VHDL configuration to simulate.
<module> ...	Specifies the name of one or more top-level Verilog modules to be simulated. Optional.

- <entity> [(<architecture>)] Specifies the name of the top-level VHDL entity to be simulated. Optional. The entity may have an architecture optionally specified; if omitted the last architecture compiled for the specified entity is simulated. An entity is not valid if a configuration is specified.¹
- <optimized_design_name> Specifies the name of an optimized design. See the [vopt](#) command. Optional.

1. Most UNIX shells require arguments containing () to be single-quoted to prevent special parsing by the shell. See the examples below.

- <MPF_file_name>
Opens the specified project. Optional.
- <WLF_file_name>
Opens the specified dataset. Optional. When you open a WLF file using the following command:

```
vsim test.wlf
```

The default behavior is to not automatically load any signals into the Wave window. You can change this behavior, such that the Wave window contains all signals in the design, by setting the preference PrefWave(OpenLogAutoAddWave) to 1 (true).

- <text_file_name>
Opens the specified text file in a Source window. Optional.

Examples

- Invoke **vsim** on the entity *cpu* and assigns values to the generic parameters *edge* and *VCC*.

```
vsim -gedge='low high' -gVCC=4.75 cpu
```

If working within the ModelSim GUI, you would enter the command as follows:

```
vsim {-gedge="low high"} -gVCC=4.75 cpu
```

Instruct ModelSim to view the results of a previous simulation run stored in the WLF file *sim2.wlf*. The simulation is displayed as a dataset named *test*. Use the **-wlf** option to specify the name of the WLF file to create if you plan to create many files for later viewing.

```
vsim -view test=sim2.wlf
```

For example:

```
vsim -wlf my_design.i01 my_asic structure
vsim -wlf my_design.i02 my_asic structure
```

Annotate instance */top/u1* using the minimum timing from the SDF file *myasic.sdf*.

```
vsim -sdfmin /top/u1=myasic.sdf
```

Use multiple switches to annotate multiple instances:

```
vsim -sdfmin /top/u1=sdf1 -sdfmin /top/u2=sdf2 top
```

- This example searches the libraries *mylib* for *top(only)* and *gatelib* for *cache_set*. If the design units are not found, the search continues to the work library. Specification of the architecture (*only*) is optional.

```
vsim 'mylib.top(only)' gatelib.cache_set
```

- Invoke **vsim** on *test_counter* and run the simulation until a break event, then quit when it encounters a \$finish task.

```
vsim -do "set PrefMain(forceQuit) 1; run -all" work.test_counter
```

vsim<info>

The **vsim<info>** commands return information about the current vsim executable.

- **vsimAuth**
Returns the authorization level (PE/SE, VHDL/Verilog/PLUS).
- **vsimDate**
Returns the date the executable was built, such as "Apr 10 2000".
- **vsimId**
Returns the identifying string, such as "ModelSim 6.1".
- **vsimVersion**
Returns the version as used by the licensing tools, such as "1999.04".
- **vsimVersionString**
Returns the full vsim version string.

This same information can be obtained using the **-version** argument of the [vsim](#) command.

vsim_break

Stop (interrupt) the current simulation before it runs to completion. To stop a simulation and then resume it, use this command in conjunction with **run -continue**.

Syntax

```
vsim_break
```

Arguments

None.

Example

- Interrupt a simulation, then restart it from the point of interruption.

```
vsim_break  
run -continue
```

vsource

This command specifies an alternative file to use for the current source file.

This command is used when the current source file has been moved. The alternative source mapping exists for the current simulation only.

Syntax

```
vsource [<filename>]
```

Arguments

- <filename>
(optional) Specifies a relative or full pathname. If filename is omitted, the source file for the current design context is displayed.

Examples

```
vsource design.vhd  
vsource /old/design.vhd
```

wave

A number of commands are available to manipulate and report on the Wave window.

The following tables summarize the available options for manipulating cursors, for zooming, and for adjusting the wave display view in the Wave window:

Table 2-13. Wave Window Commands for Cursor

Cursor Commands	Description
wave cursor active	Sets the active cursor to the specified cursor or, if no cursor is specified, reports the active cursor
wave cursor add	Adds a new cursor at specified time and returns the number of the newly added cursor
wave cursor configure	Sets or reports values for the specified cursor
wave cursor delete	Deletes the specified cursor or, if no cursor is specified, the active cursor
wave cursor see	Positions the wave display such that the specified or active cursor appears at the specified percent from the left edge of the display – 0% is the left edge, 100% is the right edge.
wave cursor time	Moves or reports the time of the specified cursor or, if no cursor is specified, the time of the active cursor

Table 2-14. Wave Window Commands for Expanded Time Display

Display view Commands	Description
wave expand mode	Selects the expanded time display mode: Delta Time, Event Time, or off.
wave expand all	Expands simulation time into delta time steps if Delta Time mode is currently selected (WLF CollapseMode = 1) or into event time steps if Event Time mode is currently selected (WLF CollapseMode = 0) over the full range of the simulation from time 0 to the current time.
wave expand cursor	Expands simulation time into delta time steps if Delta Time mode is currently selected (WLF CollapseMode = 1) or into event time steps if Event Time mode is currently selected (WLF CollapseMode = 0) at the simulation time of the active cursor.
wave expand range	Expands simulation time into delta time steps if Delta Time mode is currently selected (WLF CollapseMode = 1) or into event time steps if Event Time mode is currently selected (WLF CollapseMode = 0) over a time range specified by a start time and an end time.

Table 2-14. Wave Window Commands for Expanded Time Display (cont.)

Display view Commands	Description
wave collapse all	Collapses simulation time over the full range of the simulation from time 0 to the current time.
wave collapse cursor	Collapses simulation time at the time of the active cursor.
wave collapse range	Collapses simulation time over a specific simulation time range.

Table 2-15. Wave Window Commands for Controlling Display

Display view Commands	Description
wave interrupt	Immediately stops wave window drawing
wave refresh	Cleans wave display and redraws waves
wave cursor see	Positions the wave display such that the specified or active cursor appears at the specified percent from the left edge of the display – 0% is the left edge, 100% is the right edge.
wave seetime	Positions the wave display such that the specified time appears at the specified percent from the left edge of the display – 0% is the left edge, 100% is the right edge.

Table 2-16. Wave Window Commands for Zooming

Zooming Commands	Description
wave zoom in	Zoom in the wave display by the specified factor. The default factor is 2.0.
wave zoom out	Zoom out the wave display by the specified factor. The default factor is 2.0.
wave zoom full	Zoom the wave display to show the full simulation time.
wave zoom last	Return to last zoom range.
wave zoom range	Sets left and right edge of wave display to the specified start time and end time. If times are not specified, reports left and right edge times.

Syntax

wave cursor active [-window <win>] [<cursor-num>]

wave cursor add [-window <win>] [-time <time>] [-name <name>] [-lock <0 |1>]

wave cursor configure [<cursor-num>] [-window <win>] [<option> [<value>]]

wave cursor delete [-window <win>] [<cursor-num>]
wave cursor see [-window <win>] [-at <percent>] [<cursor-num>]
wave cursor time [-window <win>] [-time <time>] [<cursor-num>]
wave collapse all [-window <win>]
wave collapse cursor [-window <win>] [<cursor-num>]
wave collapse range [-window <win>] <start-time> <end-time>
wave expand all [-window <win>]
wave expand cursor [-window <win>] [<cursor-num>]
wave expand mode [-window <win>] [off | deltas | events]
wave expand range [-window <win>] <start-time> <end-time>
wave interrupt [-window <win>]
wave refresh [-window <win>]
wave seetime [-window <win>] [-at <percent>] -time <time>
wave zoom in [-window <win>] [<factor>]
wave zoom out [-window <win>] [<factor>]
wave zoom full [-window <win>]
wave zoom last [-window <win>]
wave zoom range [-window <win>] [<start-time> <end-time>]

Arguments

- -at <percent>
(optional) Positions the display such that the time or cursor is the specified <percent> from the left edge of the wave display.

<percent> — Any non-negative number where the default is 50. 0 is the left edge of the 100 is the right edge.
- <cursor-num>
(optional) Specifies a cursor number. If not specified, the active cursor is used.
- <factor>
(optional) A number that specifies how much you want to zoom into or out of the wave display. Default value is 2.0.
- -lock <0 |1>
(optional) Specify the lock state of the cursor.
 - 0 — (default) Unlocked
 - 1 — Locked

- `-name <name>`
(optional) Specify the name of the cursor.
`<name>` — Any string where the default is "Cursor <n>" where <n> is the cursor number.
- `off | deltas | events`
(optional) Specifies the expanded time display mode for the Wave window. Default is off.
- `<option> [<value>]`
(optional) Specify a value for the designated option. Currently supported options are `-name`, `-time`, and `-lock`. If no option is specified, current value of all options are reported.
- `<start-time> <end-time>`
(optional) `start-time` and `end-time` are times that specify an expand, collapse, or zoom range. If neither number is specified, the command returns the current range.
- `-time <time>`
(optional) Specifies a cursor time.
`<time>` — Any positive integer.
- `-window <win>`
(optional) All commands default to the active Wave window unless this argument is used to specify a different Wave window.
`<win>` — Specifies the name of a Wave window other than the current active window.

Examples

- Either of these commands creates a zoom range with a start time of 20 ns and an end time of 100 ns.

```
wave zoom range 20ns 100ns  
wave zoom range 20 100
```

- Return the name of cursor 2:

```
wave cursor configure 2 -name
```
- Name cursor 2, "reference cursor" and return that name with:

```
wave cursor configure 2 -name {reference cursor}
```

- Return the values of all wave cursor configure options for cursor 2:

```
wave cursor configure 2
```

wave create

This command generates a waveform known only to the GUI. You can then modify the waveform interactively or with the [wave edit](#) command and use the results to drive simulation.

Refer to “[Generating Stimulus with Waveform Editor](#)” for more information.

The following table summarizes the available waveform pattern options:

Command	Description
wave create -pattern clock	Generates a clock waveform. Recommended that you specify an initial value, duty cycle, and clock period for the waveform.
wave create -pattern constant	Generates a waveform with a constant value. It is suggested that you specify a value.
wave create -pattern random	Generates a random waveform based upon a seed value. Specify the type (normal or uniform), an initial value, and a seed value. If you don't specify a seed value, Questa uses a default value of 5.
wave create -pattern repeater	Generates a waveform that repeats. Specify an initial value and pattern that repeats. You can also specify how many times the pattern repeats.
wave create -pattern counter	Generates a waveform from a counting pattern. Specify start and end values, repeat, step count, time period, and type (Binary, Gray, Johnson, OneHot, Range, and ZeroHot).
wave create -pattern none	Creates a placeholder for a waveform. Specify an object name.

Syntax

Clock patterns

```
wave create -pattern clock <object_name>  
  [-driver {freeze | deposit | driver | expectedoutput}] [-duty cycle <value>]  
  [-initialvalue <value>] [-language {vhdl | verilog}] [-period {<time><unit>}]  
  [-portmode {in | out | inout | internal}] [-range <msb lsb>] [-starttime {<time><unit>}]  
  [-endtime {<time><unit>}]
```

Constant patterns

```
wave create -pattern constant <object_name>  
  [-driver {freeze | deposit | driver | expectedoutput}] [-initialvalue <value>]  
  [-language {vhdl | verilog}] [-portmode {in | out | inout | internal}] [-range <msb lsb>]  
  [-starttime {<time><unit>}] [-endtime {<time><unit>}] [-value <value>]
```

Random patterns

```

wave create -pattern random <object_name>
  [-driver {freeze | deposit | driver | expectedoutput}] [-initialvalue <value>]
  [-language {vhdl | verilog}] [-period {<time><unit>}]
  [-portmode {in | out | inout | internal}]
  [-random_type {normal | uniform | poisson | exponential}] [-range <msb lsb>]
  [-seed <value>] [-starttime {<time><unit>}] [-endtime {<time><unit>}]

```

Repeater patterns

```

wave create -pattern repeater <object_name>
  [-driver {freeze | deposit | driver | expectedoutput}] [-initialvalue <value>]
  [-language {vhdl | verilog}] [-period {<time><unit>}]
  [-portmode {in | out | inout | internal}] [-range <msb lsb>] [-repeat {forever | never | <n>}]
  [-sequence {<val1> <val2> ...}] [-starttime {<time><unit>}] [-endtime {<time><unit>}]

```

Counter patterns

```

wave create -pattern counter <object_name>
  [-direction {up | down | upthendown | downthenup}]
  [-driver {freeze | deposit | driver | expectedoutput}] [-initialvalue <value>]
  [-language {vhdl | verilog}] [-period {<time><unit>}]
  [-portmode {in | out | inout | internal}] [-range <msb lsb>] [-repeat {forever | never | <n>}]
  [-starttime {<time><unit>}] [-endtime {<time><unit>}] [-startvalue <value>]
  [-endvalue <value>] [-step <value>]
  [-type {binary | gray | johnson | onehot | range | zerohot}]

```

No pattern

```

wave create -pattern none <object_name>

```

Arguments

- <object_name>
(required) User specified name for the waveform.
- -pattern clock | constant | random | repeater | counter | none
(required) Specifies the waveform pattern. Refer to “[Creating Waveforms from Patterns](#)” for a description of the pattern types.
 - clock — Specifies a clock pattern.
 - constant — Specifies a constant pattern.
 - random — Specifies a random pattern.
 - repeater — Specifies a repeating pattern.
 - counter — Specifies a counting pattern.
 - none — Specifies a blank pattern.

- `-direction {up | down | upthendown | downthenup}`
(optional, recommended when specifying **-pattern counter**) The direction in which the counter will increment or decrement.
 - `up` — (default) Increment only.
 - `down` — Decrement only.
 - `upthendown` — Increment then decrement.
 - `downthenup` — Decrement then increment.
- `-driver {freeze | deposit | driver | expectedoutput}`
(optional) Specifies that the signal is a driver of the specified type. Applies to waveforms created with **-portmode inout** or **-portmode internal**.
- `-duty cycle <value>`
(optional, recommended for **-pattern clock**) Specifies the duty cycle of the clock. Expressed as a percentage of the period that the clock is high.
 - `<value>` — Any integer from 0 to 100 where the default is 50.
- `-endtime {<time><unit>}`
(optional) The simulation time where the waveform will stop. If omitted, the waveform stops at 1000 simulation time units.
 - `<time>` — Specified as an integer or decimal number.
 - `<unit>` — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting `<unit>`. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If `<unit>` is specified, you must enclose `<time>` and `<unit>` within curly braces (`{ }`).
- `-endvalue <value>`
(optional, recommended when specifying **-pattern counter**) The end value for the counter. This option applies to patterns specifying **-type Range** only. All other counter patterns start from 0 and go to the maximum value for that particular signal (for example, for a 3-bit signal, the start value will be 000 and the end value will be 111).
 - `<value>` — Value must be appropriate for the type of waveform you are creating.
- `-initialvalue <value>`
(optional) The initial value for the waveform. Not applicable to counter patterns.
 - `<value>` — Value must be appropriate for the type of waveform you are creating.
- `-language {vhdl | verilog}`
(optional) Controls which language is used for the created wave.
 - `vhdl` — (default) Specifies the VHDL language.
 - `verilog` — Specifies the Verilog language.

- `-period {<time><unit>}`
 (optional, recommended for all patterns except **-constant**) Specifies the period of the signal.
 - `<time>` — Specified as an integer or decimal number. Current simulation units are the default unless specifying `<unit>`.
 - `<unit>` — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting `<unit>`. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If `<unit>` is specified, you must enclose `<time>` and `<unit>` within curly braces (`{}`).
- `-portmode {in | out | inout | internal}`
 (optional) The port type for the waveform. Useful for creating signals prior to loading a design.
 - `in` — Ports of type IN. You can also specify “input” as an alias for in.
 - `out` — Ports of type OUT. You can also specify “output” as an alias for out.
 - `inout` — Ports of type INOUT.
 - `internal` — (default) Ports of type INTERNAL.
- `-random_type {normal | uniform | poisson | exponential}`
 (optional, recommended when specifying **-pattern random**) Specifies the type of algorithm used to generate a random waveform pattern.
 - `normal` — Normal or Gaussian distribution of waveform events.
 - `uniform` — Uniform distribution of waveform events.
 - `poisson` — Poisson distribution of waveform events.
 - `exponential` — Exponential distribution of waveform events.
- `-range <msb lsb>`
 (optional) Identifies bit significance in a counter pattern.
 - `msb lsb` — Most significant bit and least significant bit. Both must be specified.
- `-repeat {forever | never | <n>}`
 (optional, recommended when specifying **-pattern repeater** or **-pattern counter**) Controls duration of pattern repetition.
 - `forever` — Repeat the pattern for as long as the simulation runs.
 - `never` — Never repeat the pattern during simulation.
 - `<n>` — Repeat the pattern `<n>` number of times where `<n>` is any positive integer.
- `-seed <value>`
 (optional, recommended when specifying **-pattern random**) Specifies a seed value for a randomly generated waveform.
 - `<value>` — Any non-negative integer where the default is 5.

- -sequence {<val1>} <val2> ...
(optional, recommended when specifying **pattern -repeater**) The set of values that you want repeated.
 <val1> — Value must be appropriate for the type of waveform you are creating.
 Multiple values are entered as a space separated list and enclosed in curly braces ({}).
- -starttime {<time><unit>}
(optional) The simulation time at which the waveform should start. If omitted, the waveform starts at 0 simulation time units.
 <time> — Specified as an integer or decimal number.
 <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}).
- -startvalue <value>
(required when specifying **-pattern counter**) The initial value of the counter. This option applies to patterns specifying **-type Range** only. All other counter patterns start from 0 and go to the maximum value for that particular signal (e.g., for a 3-bit signal, the start value will be 000 and the end value will be 111).
- -step <value>
(optional, recommended when specifying **-pattern counter**) The step by which the counter is incremented/decremented.
 <value> — Value must be appropriate for the type of waveform you are creating.
 <value> — Value must be appropriate for the type of waveform you are creating.
- -type {binary | gray | johnson | onehot | range | zerohot}
(optional) Specifies a counter format.
 binary — Specifies a binary counter.
 gray — Specifies a binary counter where two successive values differ in only one bit.
 Also known as a reflected binary counter.
 johnson — Specifies a twisted ring or Johnson counter.
 onehot — Specifies a shift counter where only one bit at a time is set to “on” (1).
 range — (default) Specifies a binary counter where the values range between **-startvalue** and **-endvalue**
 zerohot — Specifies a shift counter where only one bit at a time is set to “off” (0).
- -value <value>
(optional, recommended when specifying **-pattern constant**) Specifies a value for the constant pattern.
 <value> — Value must be appropriate for the type of waveform you are creating.

Examples

- Create a clock signal with the following default values:

```
wave create -pattern clock -period 100 -duty cycle 50 -starttime 0 -endtime 1000  
-initialvalue 0 /counter/clk
```

- Create a constant 8-bit signal vector from 0 to 1000 ns with a value of 1111 and a drive type of freeze.

```
wave create -driver freeze -pattern constant -value 1111 -range 7 0 -starttime 0ns  
-endtime 1000ns sim:/andm/v_cont2
```

Related Topics

- [wave edit](#)
- [wave modify](#)
- [“Generating Stimulus with Waveform Editor”](#)
- [“Creating Waveforms from Patterns”](#)

wave edit

The **wave edit** command modifies waveforms created with the **wave create** command.

The following table summarizes the available editing options:

Command	Description
wave edit cut	Cut part of a waveform to the clipboard
wave edit copy	Copy part of a waveform to the clipboard
wave edit paste	Paste the waveform from the clipboard
wave edit invert	Vertically flip part of a waveform
wave edit mirror	Mirror part of a waveform
wave edit insert_pulse	Insert a new edge on a waveform; doesn't affect waveform duration
wave edit delete	Delete an edge from a waveform; doesn't affect waveform duration
wave edit stretch	Move an edge by stretching the waveform
wave edit move	Move an edge without moving other edges
wave edit change_value	Change the value of part of a waveform
wave edit extend	Extend all waves
wave edit driveType	Change the driver type
wave edit undo	Undo an edit
wave edit redo	Redo a previously undone edit

Syntax

wave edit {cut | copy | paste | invert | mirror} [-end <time>] -start <time> <object_name>

wave edit insert_pulse [-duration <time>] -start <time> <object_name>

wave edit delete -time <time> <object_name>

wave edit stretch | move -backward <time> | -forward <time> -time <time> <object_name>

wave edit change_value -end <time> -start <time> <value> <object_name>

wave edit extend -extend to | by -time <time>

wave edit driveType -driver freeze | deposit | driver | expectedoutput -end <time> -start <time>

The arguments below are grouped by editing operation. Many operations share similar arguments.

Arguments for cut, copy, paste, invert, and mirror

- -end <time>

The end of the section of waveform to perform the editing operation upon, denoted by a simulation time. Optional for paste.

- -start <time>
The beginning of the section of waveform to perform the editing operation upon, denoted by a simulation time. Required.
- <object_name>
The pathname of the waveform to edit. Required.

Arguments for insert_pulse

- -duration <time>
The length of the pulse. Default is 10 time units. Optional.
- -start <time>
The time at which the new pulse should be inserted. Required.
- <object_name>
The pathname of the waveform on which you are inserting a pulse. Required.

Arguments for delete

- -time <time>
The time at which the edge to delete occurs. Required.
- <object_name>
The pathname of the waveform for which you are deleting an edge. Required.

Arguments for stretch and move

- -backward <time>
The amount to stretch or move the edge backwards in simulation time. Required if -forward <time> isn't specified.
- -forward <time>
The amount to stretch or move the edge forwards in simulation time. Required if -backward <time> isn't specified.
- -time <time>
The time at which the edge to stretch or move occurs. Required.
- <object_name>
The pathname of the waveform on which you are stretching or moving an edge. Required.

Arguments for change_value

- -end <time>
The end of the section of waveform of which you are changing the value. Required.
- -start <time>
The beginning of the section of waveform of which you are changing the value. Required.

- <value>
The new value. Must match the type of the <object_name>. Required.
- <object_name>
The pathname of the waveform on which you are changing a value. Required.

Arguments for extend

- -extend to | by
Specify whether you are extending waves to a specific time or by a certain amount of time. Required.
- -time <time>
The time to extend waves to or the amount by which to extend the waves. Required.

Arguments for driveType

- -driver freeze | deposit | driver | expectedoutput
The type of driver to which you want the specified section of the waveform changed. Required.
- -end <time>
The end of the section of waveform of which you are changing the drive type. Required.
- -start <time>
The beginning of the section of waveform of which you are changing the drive type. Required.

Arguments for undo and redo

- <number>
The number of editing operations to undo or redo. If omitted, only one editing operation is undone or redone. Optional.

See also

[wave create](#), “[Generating Stimulus with Waveform Editor](#)”

wave export

The **wave export** command creates a stimulus file from waveforms created with the **wave create** command.

Syntax

```
wave export [-designunit <name>] [-endtime <time>] -file <name>  
           -format force | vcd | vhdl | verilog [-starttime <time>]
```

Arguments

- **-designunit <name>**
The name of the design unit for which you want to export created waves. If omitted, the command exports waves from the active design unit. Optional.
- **-endtime <time>**
The simulation time at which you want to stop exporting. Required.
- **-file <name>**
The filename for the saved stimulus file. Required.
- **-format force | vcd | vhdl | verilog**
The format of the saved stimulus file. Required. The format options include:
 - force** — A Tcl script that recreates the waveforms. The file should be sourced when reloading the simulation.
 - vcd** — An extended VCD file. Load using the **-vcdstim** argument to **vsim**.
 - vhdl** — A VHDL test bench. Compile and load the file as your top-level design unit.
 - verilog** — A Verilog test bench. Compile and load the file as your top-level design unit.
- **-starttime <time>**
The simulation time at which you want to start exporting. Required.

See also

[wave create](#), [wave import](#), “[Generating Stimulus with Waveform Editor](#)”

wave import

The **wave import** command imports an extended VCD file that was created with the **wave export** command. It cannot read extended VCD file created by software other than ModelSim. Use this command to apply a VCD file as stimulus to the current simulation.

Syntax

```
wave import <VCD_file>
```

Arguments

- <VCD_file>

The name of the extended VCD file to import. Required.

See also

[wave create](#), [wave export](#), [“Generating Stimulus with Waveform Editor”](#)

wave modify

The **wave modify** command modifies waveform parameters set by a previous **wave create** command. See the ModelSim Command Reference for syntax.

Syntax

```
wave modify <wave_name> [-driver freeze | deposit | driver | expectedoutput]
    [-endtime <time>] [-initialvalue <value>] -pattern clock | random | repeater | counter | none
    [-range <MSB LSB>] [-starttime <time>]
```

```
wave modify -period <value> -duty cycle <value>
```

```
wave modify -period <value> -random_type Normal|Uniform [-seed <value>]
```

```
wave modify -period <value> -repeat forever | never | <#_of_times>
    -sequence {val1 val2 val3 ...}
```

```
wave modify -direction down | up -type Binary | Range | Johnson | OneHot | ZeroHot | Gray
    -endvalue <value> -period <value> -repeat forever | never | <#_of_times>
    -startvalue <value> -step <value>
```

Arguments for all waveforms

- <wave_name>
 The name of an existing waveform created with the **wave create** command. Required.
- -driver freeze | deposit | driver | expectedoutput
 Specifies that the signal is a driver of the specified type. Applies to signals of type inout or internal. Optional.
- -endtime <time>
 The simulation time that the waveform should stop. If omitted, the waveform stops at 1000 simulation time units. Optional.
- -initialvalue <value>
 The initial value for the waveform. Value must be appropriate for the type of waveform you are creating. Not applicable to counter patterns. Optional.
- -pattern clock | random | repeater | counter | none
 The pattern for the created waveform. Refer to “[Creating Waveforms from Patterns](#)” for a description of the pattern types. Required.
- -range <MSB LSB>
 Specifies a vector of the designated bit width. Optional.
- -starttime <time>
 The simulation time that the waveform should start. If omitted, the waveform starts at 0 simulation time units. Optional.

Arguments, clock patterns only

- -period <value>
The period of the signal. Required.
- -dutycycle <value>
The duty cycle of the clock, which is the percentage of the period that the clock is high or low. Acceptable values range from 0 to 100. Required.

Arguments, random patterns only

- -period <value>
The period after which the value should change. Required.
- -random_type Normal|Uniform
The type of random pattern to generate. Required. Default is uniform.
- -seed <value>
A seed value for the random generator. If omitted, ModelSim uses the value 5. Optional.

Arguments, repeater patterns only

- -period <value>
The period after which the value should change. Required.
- -repeat forever | never | <#_of_times>
The number of times to repeat. Required.
- -sequence {val1 val2 val3 ...}
The set of values that you want repeated. Required.

Arguments, counter patterns only

- -direction down | up
The direction which the counter should increment or decrement. Optional. The default is up.
- -type Binary | Range | Johnson | OneHot | ZeroHot | Gray
The type of counter to create. Default is Range. Optional.
- -endvalue <value>
The ending value of the counter. This option applies to Range counter patterns only. All other counter patterns start from 0 and go to the max value for that particular signal (e.g., for a 3-bit signal, the start value will be 000 and end value will be 111).
- -period <value>
The period after which the value should change. Required.
- -repeat forever | never | <#_of_times>
The number of times to repeat. Required.

- -startvalue <value>

The starting value of the counter. This option applies to Range counter patterns only. All other counter patterns start from 0 and go to the max value for that particular signal (e.g., for a 3-bit signal, the start value will be 000 and end value will be 111).

- -step <value>

The step by which the counter is incremented/decremented. Required.

See also

[wave create](#), “[Generating Stimulus with Waveform Editor](#)”

wave sort

This command sorts signals in the Wave window by name or full path name.

Syntax

```
wave sort {ascending | descending | fa | fd} [-win <window_name>]
```

Arguments

- ascending | descending | fa | fd
(required) Sort signals in one of the following orders.
 - ascending — Sort in ascending order by signal name.
 - descending — Sort in descending order by signal name.
 - fa — Sort in ascending order by the full path name.
 - da — Sort in descending order by full path name.
- -win <window_name>
(optional) Specifies an instance of the Wave window that is not the default. Otherwise, the default Wave window is used. Use the [view](#) command to change the default window.
<window_name> — Window name other than the default Wave window.

Examples

```
wave sort ascending
```


when

The **when** command instructs ModelSim to perform actions when the specified conditions are met.

For example, you can use the **when** command to break on a signal value or at a specific simulator time. Use the **nowhen** command to deactivate **when** commands.

Note



When running in full optimization mode, breakpoints can not be set. Run the design in non-optimized mode (or set +acc arguments) to enable you to set breakpoints in the design. See [Preserving Object Visibility for Debugging Purposes](#).

Syntax

```
when [[-fast] [-id <id#>] [-label <label>] {<when_condition_expression>} {<command>}]
```

Description

The **when** command uses a **when_condition_expression** to determine whether or not to perform the action. Conditions can include VHDL signals and Verilog nets and registers. The **when_condition_expression** uses a simple restricted language (that is not related to Tcl), which permits only four operators and operands that may be either HDL object names, `signame'event`, or constants. ModelSim evaluates the condition every time any object in the condition changes, hence the restrictions.

Here are some additional points to keep in mind about the **when** command:

- The **when** command creates the equivalent of a VHDL process or a Verilog always block. It does not work like a looping construct you might find in other languages such as C.
- Virtual signals, functions, regions, types, etc. cannot be used in the **when** command. Neither can simulator state variables other than \$now.
- With no arguments, **when** will list the currently active when statements and their labels (explicit or implicit).

Syntax

```
when [[-fast] [-id <id#>] [-label <label>] {<when_condition_expression>} {<command>}]
```

Embedded Commands Allowed with the -fast Argument

You can use any Tcl command as a `<command>`, along with any of the following **vsim** commands:

- bp, bd
- change

- disablebp, enablebp
- echo
- examine
- force, noforce
- log, nolog
- stop
- when, nowhen

Embedded Commands Not Allowed with the -fast Argument

- Any do commands
- Any Tk commands or widgets
- References to U/I state variables or tcl variables
- Virtual signals, functions, or types

Using Global Tcl Variables with the -fast Argument

Embedded commands that use global Tcl variables for passing a state between the when command and the user interface need to declare the state using the Tcl uivar command. For example, the variable `i` below is visible in the GUI. From the command prompt, you can display it (by entering `echo $i`) or modify it (for example, by entering `set i 25`).

```
set i 10
when -fast {clk == '0'} {
    uivar i
    set i [expr {$i - 1}]
    if {$i <= 0} {
        force reset 1 0, 0 250
    }
}
when -fast {reset == '0'} {
    uivar i
    set i 10
}
```

Additional Restrictions on the -fast Argument

Accessing channels (such as files, pipes, sockets) that were opened outside of the embedded command will not work. For example:

```
set fp [open mylog.txt w]
when -fast {bus} {
    puts $fp "bus change: [examine bus]"
}
```

The channel that `$fp` refers to is not available in the simulator, only in the user interface. Even using the `uivar` command does not work here because the value of `$fp` has no meaning in the context of the `-fast` argument.

The following method of rewriting this example opens the channel, writes to it, then closes it within the **when** command:

```
when -fast {bus} {
    set fp [open mylog.txt a]
    puts $fp "bus change: [examine bus]"
    close $fp
}
```

The following example is a little more sophisticated method of doing the same thing:

```
when -fast {$now == 0ns} {
    set fp [open mylog.txt w]
}
when -fast {bus} {
    puts $fp "bus change: [examine bus]"
}
when -fast {$now == 1000ns} {
    close $fp
}
```

The general principle is that any embedded command done using the `-fast` argument is global to all other commands used with the `-fast` argument. Here, `{ $now == 0ns }` is a way to define Tcl processes that the `-fast` commands can use. These processes have the same restrictions that when bodies have, but the advantage is again speed as a proc will tend to execute faster than code in the when body itself.

It is recommended not to use virtual signals and expressions.

Arguments

- `-fast`
Causes the embedded `<command>` to execute within the simulation kernel, which provides faster execution and reduces impact on simulation runtime performance. Optional. Limitations on using the `-fast` argument are described above (in “[Embedded Commands Not Allowed with the -fast Argument](#)”). Disallowed commands still work, but they slow down the simulation.
- `-label <label>`
Used to identify individual **when** commands. Optional.
- `-id <id#>`
Attempts to assign this id number to the when command. Optional. If the id number you specify is already used, ModelSim will return an error.

Note



Id numbers for when commands are assigned from the same pool as those used for the `bp` command. So even if you have not specified a given id number for a when command, that number may still be used for a breakpoint.

- {<when_condition_expression>}

Specifies the conditions to be met for the specified <command> to be executed. Required if a command is specified. The condition is evaluated in the simulator kernel and can be an object name, in which case the curly braces can be omitted. The command will be executed when the object changes value. The condition can be an expression with these operators:

Name	Operator
equals	==, =
not equal	!=, /=
greater than	>
less than	<
greater than or equal	>=
less than or equal	<=
AND	&&, AND
OR	, OR

The operands may be object names, `signame'event`, or constants. Subexpressions in parentheses are permitted. The command will be executed when the expression is evaluated as TRUE or 1.

The formal BNF syntax is:

```
condition ::= Name | { expression }  
  
expression ::= expression AND relation  
              | expression OR relation  
              | relation  
relation ::= Name = Literal  
            | Name /= Literal  
            | Name ' EVENT  
            | ( expression )  
Literal ::= '<char>' | "<bitstring>" | <bitstring>
```

The "=" operator can occur only between a Name and a Literal. This means that you cannot compare the value of two signals, i.e., `Name = Name` is not possible.

Tcl variables can be used in the condition expression but you must replace the curly braces (`{}`) with double quotes (`"`). This works like a macro substitution where the Tcl variables are evaluated once and the result is then evaluated as the when condition. Condition expressions are evaluated in the **vsim** kernel, which knows nothing about Tcl variables. That's why the condition expression must be evaluated in the GUI before it is sent to the **vsim** kernel. See below for an example of using a Tcl variable.

The ">", "<", ">=", and "<=" operators are the standard ones for vector types, not the overloaded operators in the `std_logic_1164` package. This may cause unexpected results when comparing objects that contain values other than 1 and 0. ModelSim does a lexical comparison (position number) for values other than 1 and 0. For example:

```
0000 < 1111 ## This evaluates to true
H000 < 1111 ## This evaluates to false
001X >= 0010 ## This also evaluates to false
```

- {<command>}

The command(s) for this argument are evaluated by the Tcl interpreter within the ModelSim GUI. Any ModelSim or Tcl command or series of commands are valid with one exception—the **run** command cannot be used with the **when** command. Required if a when expression is specified. The command sequence usually contains a **stop** command that sets a flag to break the simulation run after the command sequence is completed. Multiple-line commands can be used.

Note



If you want to stop the simulation using a **when** command, you must use a **stop** command within your when statement. DO NOT use an **exit** command or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated.

Examples

- The **when** command below instructs the simulator to display the value of object *c* in binary format when there is a clock event, the clock is 1, and the value of *b* is 01100111. Finally, the command tells ModelSim to stop.

```
when -label when1 {clk'event and clk='1' and b = "01100111"} {
    echo "Signal c is [exa -bin c]"
    stop
}
```

- The commands below show an example of using a Tcl variable within a **when** command. Note that the curly braces ({}) have been replaced with double quotes ("").

```
set clkb_path /tb/ps/dprb_0/udprb/ucar_reg/uint_ram/clkb;
when -label when1 "$clkb_path'event and $clkb_path = '1'" {
    echo "Detected Clk edge at path $clkb_path"
}
```

- The **when** command below is labeled *a* and will cause ModelSim to echo the message “b changed” whenever the value of the object *b* changes.

```
when -label a b {echo "b changed"}
```

- The multi-line **when** command below does not use a label and has two conditions. When the conditions are met, ModelSim runs an **echo** command and a **stop** command.

```
when {b = 1
    and c /= 0 } {
    echo "b is 1 and c is not 0"
    stop
}
```

- In the example below, for the declaration "wire [15:0] a;", the **when** command will activate when the selected bits match a 7:

```
when {a(3:1) = 3'h7} {echo "matched at time " $now}
```

- If you encounter a vectored net caused by optimizing with **vopt**, use the 'event qualifier to prevent the command from falsely evaluating when unrelated bits of 'a' change:

```
when {a(3:1) = 3'h7 and a(3:1)'event} {echo "matched at time " $now}
```

- In the example below, we want to sample the values of the address and data bus on the first falling edge of *clk* after *sstrb* has gone high.

```
# ::strobe is our state variable
set ::strobe Zero
# This signal breakpoint only fires when sstrb changes to a '1'
when -label checkStrobe {/top/sstrb == '1'} {
  # Our state Zero condition has been met, move to state One
  set ::strobe One
}
# This signal breakpoint fires each time clk goes to '0'
when {/top/clk == '0'} {
  if {::$strobe eq "One"} {
    # Our state One condition has been met
    # Sample the busses
    echo Sample paddr=[examine -hex /top/paddr] :: sdata=[examine
-hex
  /top/sdata]
    # reset our state variable until next rising edge of sstrb
(back to
state Zero)
    set ::strobe Zero
  }
}
```

Ending the simulation with the stop command

Batch mode simulations are often structured as "run until condition X is true," rather than "run for X time" simulations. The multi-line **when** command (shown below) sets a done condition, and ModelSim runs an **echo** command and a **stop** command when the condition is reached.

The simulation will not stop (even if a **quit -f** command is used) unless you enter a **stop** command. To exit the simulation and quit ModelSim, use an approach like the following:

```
onbreak {resume}
when {/done_condition == '1'} {
  echo "End condition reached"
  if [batch_mode] {
    set DoneConditionReached 1
    stop
  }
}
run 1000 us
if {$DoneConditionReached == 1} {
  quit -f
}
```

This example stops 100ns after a signal transition:

```
when {a = 1} {  
  # If the 100ns delay is already set then let it go.  
  if {[when -label a_100] == ""} {  
    when -label a_100 { $now = 100 } {  
      # delete this breakpoint then stop  
      nowhen a_100  
      stop  
    }  
  }  
}
```

Time-based breakpoints

You can build time-based breakpoints into a **when** statement with the following syntax.

For absolute time (indicated by @) use:

```
when {$now = @1750 ns} {stop}
```

You can also use:

```
when {errorFlag = '1' OR $now = 2 ms} {stop}
```

This example adds 2 ms to the simulation time at which the **when** statement is first evaluated, then stops. The white space between the value and time unit is required for the time unit to be understood by the simulator.

You can also use variables, as shown in the following example:

```
set time 1000  
when "\$now = $time" {stop}
```

The quotes instruct Tcl to expand the variables before calling the command. So, the **when** command sees:

```
when "$now = 1000" stop
```

Note that "\$now" has the '\$' escaped. This prevents Tcl from expanding the variable, because if it did, you would get:

```
when "0 = 1000" stop
```

See also

[bp](#), [disablebp](#), [enablebp](#), [nowhen](#)

where

The **where** command displays information about the system environment. This command is useful for debugging problems where ModelSim cannot find the required libraries or support files.

The command displays two results on consecutive lines:

- current directory

This is the current directory that ModelSim was invoked from, or that was specified on the ModelSim command line.

- current project file

This is the *.mpf* file ModelSim is using. All library mappings are taken from here when a project is open. If the design is not loaded through a project, this line displays the *modelsim.ini* file in the current directory.

Syntax

where

Arguments

- None.

Examples

- Design is loaded through a project:

```
VSIM> where
# Current directory is: D:\Client
# Project is: D:/Client/monproj.mpf
```

- Design is loaded with no project (indicates the *modelsim.ini* file is under the *mydesign* directory):

```
VSIM> where
# Current directory is: C:\Client\testcase\mydesign
# Project is: modelsim.ini
```


wlf2log

The **wlf2log** command translates a ModelSim WLF file (*vsim.wlf*) to a QuickSim II logfile.

The command reads the *vsim.wlf* WLF file generated by the **add list**, **add wave**, or **log** commands in the simulator and converts it to the QuickSim II logfile format.

Note



This command should be invoked only after you have stopped the simulation using **quit -sim** or **dataset close sim**.

Syntax

```
wlf2log [-bits] [-fullname] [-help] [-inout] [-input] [-internal] [-l <instance_path>] [-lower] [-o <outfile>] [-output] [-quiet] <wlffile>
```

Arguments

- **-bits**
Forces vector nets to be split into 1-bit wide nets in the log file. Optional.
- **-fullname**
Shows the full hierarchical pathname when displaying signal names. Optional.
- **-help**
Displays a list of command options with a brief description for each. Optional.
- **-inout**
Lists only the inout ports. Optional. This may be combined with the **-input**, **-output**, or **-internal** switches.
- **-input**
Lists only the input ports. Optional. This may be combined with the **-output**, **-inout**, or **-internal** switches.
- **-internal**
Lists only the internal signals. Optional. This may be combined with the **-input**, **-output**, or **-inout** switches.
- **-l <instance_path>**
Lists the signals at or below the specified HDL instance path within the design hierarchy. Optional.
- **-lower**
Shows all logged signals in the hierarchy. Optional. When invoked without the **-lower** switch, only the top-level signals are displayed.

- **-o <outfile>**
Directs the output to be written to the file specified by <outfile>. Optional. The default destination for the logfile is standard out.
- **-output**
Lists only the output ports. Optional. This may be combined with the **-input**, **-inout**, or **-internal** switches.
- **-quiet**
Disables error message reporting. Optional.
- **<wlffile>**
Specifies the ModelSim WLF file that you are converting. Required.

Additional information for QuickSim II users

In some cases your original QuickHDL/ModelSim simulation results (in your *vsim.wlf* file) may contain signal values that do not directly correspond to `qsim_12state` values. The resulting QuickSim II logfile generated by **wlf2log** may contain state values that are surrounded by single quotes, e.g. '0' and '1'. To make this logfile compatible with QuickSim models (that expect `qsim_12state`) you need to use a QuickSim II function named `$convert_wdb()`.

This function was created to convert logfiles resulting from VHDL simulation that used `std_logic` and `std_ulogic` since these data types do not correlate to QuickSim's 12 simulation states. Other VHDL data types such as `qsim_state` or `bit` (2 state) do not require conversion as they are directly compatible with `qsim_12state` QuickSim II Waveform Databases (WDB).

The following procedure can be used to convert a **wlf2log**-generated logfile into a compatible QuickSim WDB. The procedure below shows how to convert the logfile while loaded into memory in QuickSim II.

1. Load the logfile (the output from **wlf2log**) into a WDB other than "forces". "Forces" is the default WDB, so you need to choose a unique name for the WDB when loading the logfile (for example, "fred").
2. Enter the following at the command prompt from within QuickSim:

```
$convert_wdb("fred",0)
```

The first argument, which is "fred", is the name of the new WDB to be created. The second argument, which is 0, specifies the type of conversion. At this time only one type of conversion is supported. The value 0 specifies to convert `std_logic` or `std_ulogic` into `qsim_12state`.

3. Do a `connect_wdb` (either through the pulldown menus, the "Connect WDB" palette icon under "Stimulus", or a function invocation). You specify the name of the WDB that you originally loaded the logfile into (in this case, "fred").

At this point you can issue the "run" command and the stimulus in the converted logfile will be applied. Before exiting the simulation you should save the new WDB ("fred") as a WDB or

logfile so that it can be loaded again in the future. The new WDB or logfile will contain the correct `qsim_12state` values eliminating the need to re-use `$convert_wdb()`.

wlf2vcd

The **wlf2vcd** command translates a ModelSim WLF file to a standard VCD file. Complex data types that are unsupported in the VCD standard (records, memories, etc.) are not converted.

Note



This command should be invoked only after you have stopped the simulation using **quit -sim** or **dataset close sim**.

Syntax

```
wlf2vcd [-help] [-o <outfile>] [-quiet] <wlf file>
```

Arguments

- **-help**
Displays a list of command options with a brief description for each. Optional.
- **-o <outfile>**
Specifies a filename for the output. By default, the VCD output goes to stdout. Optional.
- **-quiet**
Disables warning messages that are produced when an unsupported type (e.g., records) is encountered in the WLF file. Optional.
- **<wlf file>**
Specifies the ModelSim WLF file that you are converting. Required.

wlfman

This command allows you to get information about and manipulate saved WLF files.

The command performs four functions depending on which mode you use:

- **wlfman info** returns file information, resolution, versions, and so forth about the source WLF file.
- **wlfman items** generates a list of HDL objects (i.e., signals) from the source WLF file and outputs it to stdout. When redirected to a file, the output is called an `object_list_file`, and it can be read in by **wlfman filter**. The `object_list_file` is a list of objects, one per line. Comments start with a '#' and continue to the end of the line. Wildcards are legal in the leaf portion of the name. Here is an example:

```
/top/foo      # signal foo
/top/u1/*     # all signals under u1
/top/u1       # same as line above
-r /top/u2    # recursively, all signals under u2
```

Note that you can produce these files from scratch but be careful with syntax. **wlfman items** always creates a legal `object_list_file`.

- **wlfman filter** reads in a WLF file and, optionally, an `object_list_file`, and writes a new WLF file containing filtered information from those sources. You determine the filtered information with the arguments you specify.
- **wlfman profile** generates a report of the estimated percentage of file space that each signal is taking in the specified WLF file. This command can identify signals that account for a large percentage of the WLF file size (e.g., a logged memory that uses a zero-delay integer loop to initialize the memory). You may be able to drastically reduce WLF file size by not logging those signals.
- **wlfman merge** combines two WLF files with different signals into one WLF file. It *does not* combine wlf files containing the same signals at different runtime ranges (i.e., `mixedhdl_0ns_100ns.wlf` & `mixedhdl_100ns_200ns.wlf`).
- **wlfman optimize** copies the data from the WLF file to the output WLF file, adding or replacing the indexing and optimization information.

The different modes are intended to be used together. For example, you might run **wlfman profile** and identify a signal that accounts for 50% of the WLF file size. If you don't actually need that signal, you can then run **wlfman filter** to remove it from the WLF file.

Syntax

```
wlfman info <source_wlffile> [-v]
```

```
wlfman items <source_wlffile> [-n] [-v]
```

```
wlfman filter -o <out_wlffile> <source_wlffile> [-begin <time>] [-end <time>]
[-compress | -nocompress] [-f <object_list_file>] [-index | -noindex] [-r <object>]
[-opt | -noopt] [-s <symbol>] [-t <resolution>]
```

wlfman profile **<source_wlffile>** [-rank] [-top <number>]

wlfman merge **-o <out_wlffile>** [**<wlffile1> <wlffile2> ...**] [-compress | -nocompress]
[-index | -noindex] [-opt | -noopt]

wlfman optimize **-o <out_wlffile> <source_wlffile>** [-opt | -noopt]

Arguments

- **-o <out_wlffile>**
(required) Specifies the name of the output WLF file. The output WLF file will contain all objects specified by the preceding arguments. Output WLF files are always written in the latest WLF version regardless of the source WLF file version.
- **<source_wlffile>**
(required) Specifies the WLF file from which you want information.
- **<wlffile1> <wlffile2> ...**
(required) Specifies the WLF files whose objects you want to copy into one WLF file. Specified as a space separated list.
- **-begin <time>**
(optional) Specifies the simulation time at which to start reading information from the source WLF file where the default is to include the entire length of time recorded in **<source_wlffile>**.
- **-compress | -nocompress**
(optional) Controls compression of the output WLF file.
-compress — Enables compression. (default)
-nocompress — Disables compression.
- **-end <time>**
(optional) Specifies the simulation time at which you want to end reading information from **<source_wlffile>**.
- **-f <object_list_file>**
(optional) Specifies an **object_list_file** created by **wlfman items** to include in **<outwolffile>**.
- **-index | -noindex**
(optional) Controls indexing when writing the output WLF file. Indexing makes viewing wave data faster, however performance during optimization will be slower because indexing and optimization require significant memory and CPU resources. Disabling indexing makes viewing wave data slower unless the display is near the start of the WLF file. Disabling indexing also disables optimization of the WLF file but may provide a significant performance boost when archiving WLF files. Indexing and optimization information can be added back to the file using the **wlfman optimize** command.

-
- index — Enables indexing. (default)
 - noindex — Disables indexing and optimization.
 - -n
(optional) Lists regions only (no signals).
 - -opt | -noopt
(optional) Controls optimization of the output WLF file.
 - opt — Enables WLF file optimization. (default)
 - noopt — Disables WLF file optimization.
 - -r <object>
(optional) Specifies an object (region) to recursively include in the output. If <object> is a signal, the output would be the same as using -s.
 - -rank
(optional) Sorts the **wlfman profile** report by percentage of the total file space used by each signal.
 - -s <symbol>
(optional) Specifies an object to include in the output. By default all objects are included.
 - -t <resolution>
(optional) Specifies the time resolution of the new WLF file. By default the resolution is the same as the source WLF file.
 - -top <number>
(optional) Filters the **wlfman profile** report so that only the top <number> signals in terms of file space percentage are displayed.
 - -v
(optional) Produces verbose output that lists the object type next to each object.

Examples

- The output from this command would look something like this:

```
wlfman profile -rank top_vh.wlf
```

Returns:

```
#Repeated ID #'s mean those signals share the same
#space in the wlf file.
#
# ID      Transitions   File %   Name
#-----
# 1         2192         33 %    /top_vh/pdata
# 1         2192         33 %    /top_vh/processor/data
# 1         2192         33 %    /top_vh/cache/pdata
# 1         2192         33 %    /top_vh/cache/gen__0/s/data
# 1         2192         33 %    /top_vh/cache/gen__1/s/data
# 1         2192         33 %    /top_vh/cache/gen__2/s/data
# 1         2192         33 %    /top_vh/cache/gen__3/s/data
# 2         1224         18 %    /top_vh/ptrans
# 3         1216         18 %    /top_vh/sdata
# 3         1216         18 %    /top_vh/cache/sdata
# 3         1216         18 %    /top_vh/memory/data
# 4          675         10 %    /top_vh/strans
# 5          423          6 %    /top_vh/cache/gen__3/s/data_out
# 6          135          3 %    /top_vh/paddr.
.
.
.
```

- wlfman profile -top 3 top_vh.wlf

The output from this command would look something like this:

```
# ID      Transitions   File %   Name
#-----
# 1         2192         33 %    /top_vh/pdata
# 1         2192         33 %    /top_vh/processor/data
# 1         2192         33 %    /top_vh/cache/pdata
# 1         2192         33 %    /top_vh/cache/gen__0/s/data
# 1         2192         33 %    /top_vh/cache/gen__1/s/data
# 1         2192         33 %    /top_vh/cache/gen__2/s/data
# 1         2192         33 %    /top_vh/cache/gen__3/s/data
# 2         1224         18 %    /top_vh/ptrans
# 3         1216         18 %    /top_vh/sdata
# 3         1216         18 %    /top_vh/cache/sdata
# 3         1216         18 %    /top_vh/memory/data
```

Related Topics

- [Recording Simulation Results With Datasets](#)
- [WLF File Parameter Overview](#)

wlrecover

The **wlrecover** tool attempts to "repair" WLF files that are incomplete due to a crash or the file being copied prior to completion of the simulation. You can run the tool from the VSIM> or ModelSim> prompt or from a shell.

Syntax

```
wlrecover <filename> [-force] [-q]
```

Arguments

- <filename>
Specifies the WLF file to repair. Required.
- -force
Disregards file locking and attempts to repair the file. Optional.
- -q
Hides all messages unless there is an error while repairing the file. Optional.

write cell_report

The **write cell_report** command writes to the Transcript window or to a file a list of Verilog modules which qualified for and received gate-level cell optimizations. Gate-level cell optimizations are applied at the module level, in addition to normal Verilog optimizations, to improve performance of gate-level simulations.

Syntax

```
write cell_report [-filter <number>] [-infile <filename>] [-nonopt] [[-outfile] <filename>]
```

Arguments

- **-filter <number>**
Excludes cells with instance counts fewer than <number>. Optional.
- **-infile <filename>**
Specifies a previously generated write report file to use as input. Optional. If not specified then the write report command will be run.
- **-nonopt**
Reports only non-optimized instances. Optional.
- **[-outfile] <filename>**
Writes the report to the specified output file rather than the Transcript window. Optional.

write format

The **write format** command records the names and display options of the HDL objects currently being displayed in the Analysis, List, Memory, Message Viewer, Test Browser, and Wave windows.

The **write format restart** command creates a single *.do* file that will recreate all debug windows, all file/line breakpoints, and all signal breakpoints created using the **when** command. If the **ShutdownFile** *modelsim.ini* variable is set to this *.do* filename, it will call the write format restart command upon exit.

The file created is primarily a list of **add list**, **add wave**, and **configure** commands, though a few other commands are included (see "Output" below). This file may be invoked with the **do** command to recreate the window format on a subsequent simulation run.

When you load a format file, ModelSim verifies the existence of the datasets required by that file. ModelSim displays an error message if the requisite datasets do not all exist. To force the execution of the format file even if all datasets are not present, use the **-force** switch with your **do** command. For example:

```
VSIM> do format.do -force
```

Note that this will result in error messages for signals referencing nonexistent datasets. Also, **-force** is recognized by the format file not the **do** command.

Syntax

```
write format analysis | list | memory | msgviewer | testbrowser | wave | restart [-window  
    <window_name>] <filename>
```

Arguments

- analysis | list | memory | msgviewer | testbrowser | wave | restart
Specifies that the contents of the designated window are recorded in a *.do* file, <filename>. If restart is designated, all windows and breakpoints are recorded in the *.do* file. Required.
- -window <window_name>
Specifies the List or Wave window for which you want contents recorded. Optional. Use when you have more than one instance of the List or Wave window.
- <filename>
Specifies the name of the output file where the data is to be written. Required.

Examples

- Save the current data in the List window in a file named *alu_list.do*.

```
write format list alu_list.do
```
- Save the current data in the Wave window in a file named *alu_wave.do*.

```
write format wave alu_wave.do
```

Output

- Below is an example of a saved Wave window format file.

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -format Logic /cntr_struct/ld
add wave -noupdate -format Logic /cntr_struct/rst
add wave -noupdate -format Logic /cntr_struct/clock
add wave -noupdate -format Literal /cntr_struct/d
add wave -noupdate -format Literal /cntr_struct/q
TreeUpdate [SetDefaultTree]
quietly WaveActivateNextPane
add wave -noupdate -format Logic /cntr_struct/p1
add wave -noupdate -format Logic /cntr_struct/p2
add wave -noupdate -format Logic /cntr_struct/p3
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {0 ns}
WaveRestoreZoom {0 ns} {1 us}
configure wave -namecolwidth 150
configure wave -valuecolwidth 100
configure wave -signalnamewidth 0
configure wave -justifyvalue left
```

In the example above, five signals are added with the *-noupdate* argument to the default window. The **TreeUpdate** command then refreshes all five waveforms. The second **WaveActivateNextPane** command creates a second pane which contains three signals. The **WaveRestoreCursors** command restores any cursors you set during the original simulation, and the **WaveRestoreZoom** command restores the Zoom range you set. These four commands are used only in saved Wave format files; therefore, they are not documented elsewhere.

See also

[add list](#), [add wave](#), [configure](#)

write list

The **write list** command records the contents of the most recently opened or specified List window in a list output file.

This file contains simulation data for all HDL objects displayed in the List window: VHDL signals and variables and Verilog nets and registers.

Syntax

```
write list [-events] [-window <wname>] <filename>
```

Arguments

- **-events**
Specifies to write print-on-change format. Optional. Default is tabular format.
- **-window <wname>**
Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the [view](#) command to change the default window.
- **<filename>**
Specifies the name of the output file where the data is to be written. Required.

Examples

- Save the current data in the default List window in a file named *alu.lst*.

```
write list alu.lst
```
- Save the current data in window 'list1' in a file named *group1.lst*.

```
write list -win list1 group1.lst
```

See also

[write tssi](#)

write preferences

The **write preferences** command saves the current GUI preference settings to a Tcl preference file. Settings saved include Wave, Objects, and Locals window column widths; Wave, Objects, and Locals window value justification; and Wave window signal name width.

Syntax

```
write preferences <preference file name>
```

Arguments

- <preference file name>
Specifies the name for the preference file. Optional. If the file is named *modelsim.tcl*, ModelSim will read the file each time **vsim** is invoked. To use a preference file other than *modelsim.tcl* you must specify the alternative file name with the [MODELSIM_TCL](#) environment variable.

See also

You can modify variables by editing the preference file with the ModelSim [notepad](#):

```
notepad <preference file name>
```

write report

The **write report** command prints a summary of the design being simulated including a list of all design units (VHDL configurations, entities, and packages, and Verilog modules) with the names of their source files. The summary includes a list of all source files used to compile the given design.

The Simulation Report contains the following information:

- Design Simulated — directory path of the design's top-level module
- Number of signals/nets in the design
- Number of processes in the design
- Simulator Parameters, including:
 - Current directory
 - Project file directory
 - Simulation time resolution
- List of design units used, including:
 - Module name
 - Architecture, if applicable
 - Library directory
 - Source file
 - Timescale
 - Occurrences

Syntax

```
write report [-capacity [-l | -s] [-assertions | -classes | -cvg | -qdas | -solver]] |  
[-solver] |  
[-l | -s] [-tcl] [<filename>]
```

Arguments

- <filename>
Specifies the name of the output file where the data is to be written. Optional. If the <filename> is omitted, the report is written to the Transcript window.
- -capacity
Reports data on memory usage of various types of SystemVerilog constructs in the design. Optional. ModelSim collects memory usage data for assertions, classes, covergroups, dynamic objects, and the solver. Each of these design object types has a switch that you can

specify along with `-capacity` in order to display its memory data. To display memory data for all object types, specify `-capacity -l`.

- `-assertions`
Reports memory usage data for SystemVerilog assertions and cover directives.
- `-classes`
Reports memory usage data for the current number of objects allocated, the current memory allocated for class object, the peak memory allocated and peak time.
- `-cvg`
Reports memory usage data for the number of covergroups, cross, bins and memory allocated.
- `-qdas`
Reports memory usage data for queues, dynamic arrays, and associative arrays.
- `-solver`
Reports memory usage data for calls to `randomize()` and memory usage.
- `-l`
Generates more detailed information about the design, including a list of sparse memories or the memory capacity for all object types. Default.
- `-s`
Generates a short list of design information. Optional.
- `-solver`
Summarizes information about the memory allocated by the constraint solver and returns statistics for every SystemVerilog `randomize()` call site by filename and line number. Optional.
- `-tcl`
Generates a Tcl list of design unit information. Optional. This argument cannot be used with a filename.

Examples

- Save information about the current design in a file named `alu_rpt.txt`.

```
write report alu_rpt.txt
```
- Display a short list of information regarding the memory capacity for covergroups in the design during the simulation so far.

```
write report -capacity -s cvg
```
- Display information on all of the calls to `randomize()` made during simulation so far, along with the memory usage of those calls, number of calls, and callsite information.


```
write report -capacity -solver
```

Example Output

```
##
## SIMULATION REPORT           Generated on Mon Aug 10 12:56:15 2009
##
##
## Design simulated: <directory>\work.top(fast)
## Number of signals/nets in design: 89
## Number of processes in design: 74
##
## Simulator Parameters:
##
##   Current directory: <directory>\
##   Project file: <directory>\win32\..\modelsim.ini
##   Simulation time resolution: 1ns
##
## List of Design units used:
##
##   Module: top
##   Architecture: fast
##   Library: <directory>\work
##   Source File: top.v
##   Timescale: 1ns / 1ns
##   Occurrences: 1
##
##   Module: proc
##   Architecture: fast
##   Library: <directory>\work
##   Source File: proc.v
##   Timescale: 1ns / 1ns
##   Occurrences: 1
##
...

```

write timing

The **write timing** command displays path delays and timing check limits, unadjusted for delay net delays, for the specified instance.

When the **write timing** command reports interconnect delays on a Verilog module instance you will see either MIPDs (Module Input Port Delays) or MITDs (Module Transport Port Delays) reported. If you specify either the **+multisource_int_delays** or the **+transport_int_delays** switch with the `vsim` command, INTERCONNECT delays will be reported as MITDs. Otherwise they will be reported as MIPDs. An MIPD report may look like the following:

```
# /top/u1: [mymod:src/5/test.v(18)]
#   MIPD(s):
#     Port clk_in: (6, 6, 6)
```

An MITD report may look like the following:

```
# /top/u1: [mymod:src/5/test.v(18)]
#   MITDs to port clk_in:
#     From port /top/p/y = (6)
```

Syntax

```
write timing [-recursive] [-file <filename>] [<instance_name1>...<instance_nameN>]
[-simvalues]
```

Arguments

- **-recursive**
Generates timing information for the specified instance and all instances underneath it in the design hierarchy. Optional.
- **-file <filename>**
Specifies the name of the output file where the data is to be written. Optional. If the **-file** argument is omitted, timing information is written to the Transcript window.
- **<instance_name1>...<instance_nameN>**
The name(s) of the instance(s) for which timing information will be written. Required.
- **-simvalues**
Displays optimization-adjusted values for delay net delays. Optional.

Examples

- Write timing about `/top/u1` and all instances underneath it in the hierarchy to the file `timing.txt`.

```
write timing -r -f timing.txt /top/u1
```

- Write timing information about the designated instances to the Transcript window.

```
write timing /top/u1 /top/u2 /top/u3 /top/u8
```

write transcript

The **write transcript** command writes the contents of the Transcript window to the specified file. The resulting file can be used to replay the transcribed commands as a DO file (macro).

The command cannot be used in batch mode. In batch mode use the standard Transcript file or redirect stdout.

Syntax

```
write transcript [<filename>]
```

Arguments

- <filename>
Specifies the name of the output file where the data is to be written. Optional. If the <filename> is omitted, the transcript is written to a file named *transcript*.

See also

[do](#)

write tssi

The **write tssi** command records the contents of the default or specified List window in a "TSSI format" file.

The file contains simulation data for all HDL objects displayed in the List window that can be converted to TSSI format (VHDL signals and Verilog nets). A signal definition file is also generated.

The List window needs to be using symbolic radix in order for **write tssi** to produce useful output.

Syntax

```
write tssi [-window <wname>] <filename>
```

Arguments

- `-window <wname>`
Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the [view](#) command to change the default window.
- `<filename>`
Specifies the name of the output file where the data is to be written. Required.

Description

If the `<filename>` has a file extension (e.g., *listfile.lst*), then the definition file is given the same file name with the extension *.def* (e.g., *listfile.def*). The values in the listfile are produced in the same order that they appear in the List window. The directionality is determined from the port type if the object is a port, otherwise it is assumed to be bidirectional (mode INOUT).

Objects that can be converted to SEF are VHDL enumerations with 255 or fewer elements and Verilog nets. The enumeration values U, X, 0, 1, Z, W, L, H and - (the enumeration values defined in the IEEE Standard 1164 **std_ulogic** enumeration) are converted to SEF values according to the table below. Other values are converted to a question mark (?) and cause an error message. Though the **write tssi** command was developed for use with **std_ulogic**, any signal which uses only the values defined for **std_ulogic** (including the VHDL standard type **bit**) will be converted.

std_ulogic State Characters	SEF State Characters		
	Input	Output	Bidirectional
U	N	X	?
X	N	X	?
0	D	L	0
1	U	H	1
Z	Z	T	F

std_ulogic State Characters	SEF State Characters		
	Input	Output	Bidirectional
W	N	X	?
L	D	L	0
H	U	H	1
-	N	X	?

Bidirectional logic values are not converted because only the resolved value is available. The TSSI TDS ASCII In Converter and ASCII Out Converter can be used to resolve the directionality of the signal and to determine the proper forcing or expected value on the port. Lowercase values x, z, w, l, and h are converted to the same values as the corresponding capitalized values. Any other values will cause an error message to be generated the first time an invalid value is detected on a signal, and the value will be converted to a question mark (?).

Note



The TDS ASCII In Converter and ASCII Out Converter are part of the TDS software. ModelSim outputs a vector file, and TSSI tools determine whether the bidirectional signals are driving or not.

See also

[tssi2mti](#)

write wave

The **write wave** command records the contents of the most currently opened or specified Wave window in PostScript format.

The output file can then be printed on a PostScript printer.

Syntax

```
write wave [-window <wname>] [-width <real_num>] [-height <real_num>]  
          [-margin <real_num>] [-start <time>] [-end <time>] [-perpage <time>] [-landscape]  
          [-portrait] <filename>
```

Arguments

- **-window <wname>**
Specifies an instance of the Wave window that is not the default. Optional. Otherwise, the default Wave window is used. Use the [view](#) command to change the default window.
- **-width <real_num>**
Specifies the paper width in inches. Optional. Default is 8.5.
- **-height <real_num>**
Specifies the paper height in inches. Optional. Default is 11.0.
- **-margin <real_num>**
Specifies the margin in inches. Optional. Default is 0.5.
- **-start <time>**
Specifies the start time (on the waveform timescale) to be written. Optional.
- **-end <time>**
Specifies the end time (on the waveform timescale) to be written. Optional.
- **-perpage <time>**
Specifies the time width per page of output. Optional.
- **-landscape**
Use landscape (horizontal) orientation. Optional. This is the default orientation.
- **-portrait**
Use portrait (vertical) orientation. Optional. The default is landscape (horizontal).
- **<filename>**
Specifies the name of the PostScript output file. Required.

Examples

- Save the current data in the Wave window in a file named *alu.ps*.

```
write wave alu.ps
```

- Save the current data in window 'wave2' in a file named *group2.ps*.

```
write wave -win wave2 group2.ps
```

- Write two separate pages to *top.ps*. The first page contains data from 600ns to 700ns, and the second page contains data from 701ns to 800ns.

```
write wave -start 600ns -end 800ns -perpage 100ns top.ps
```

To make the job of creating a PostScript waveform output file easier, use the **File > Print Postscript** menu selection in the Wave window.

xml2ucdb

The **xml2ucdb** is a utility used to convert an XML test plan file to a *.ucdb* file. The configuration settings for this utility are read automatically from the *xml2ucdb.ini* file, located in *<install_dir>/vm_src/* directory.

The settings specified by this command override any settings in the *xml2ucdb.ini* file (see “[xml2ucdb.ini Configuration File](#)”).

For information about the XML language, see the “XML 1.0 Specification” available on the web.

Syntax

```
xml2ucdb [<options>] <XML_filename> [<ucdb_filename>]
```

Where <options> are:

```
[-help]  
[-debug] [-verbose] [-version]  
[-viewtags] [-viewall] [-formatlist] [-format <format>]  
[-excelsheet <sheet_name>] [-dofilename <file>] [-ucdbfilename <file>]  
[-inherit]  
[-searchpath <path/to/XML_input>]  
[-stylesheet <file>]  
[-tagseparators <str>] [-starttags <tags>] [-stoptags <tags>] [-excludetags <tags>]  
[-sectiontags <tag>] [-datatags <tag>] [-titletag <tag>]  
[-descriptiontag <tag>] [-goaltag <tags>] [-weighttag <tags>] [-linktag <tag>]  
[-modelsimini <ini_filepath>]  
[-typeattr <name>] [-linkattr <tag>] [-datafields <str>] [-datalabels <str>]  
[-autonumber | -noautonumber]  
[-startsection <num>] [-startstoring <num>]  
[-root <str>] [-title <str>] [-tagprefix <str>] [-sectionprefix <str>]
```

Arguments

- -autonumber | -noautonumber

Enables (-autonumber) or disables (-noautonumber) the automatic generation of testitem numbers from section tags. Optional. By default, the autonumbering is disabled. Use -autonumber to enable it. If you override the default off setting by enabling autonumbering in a custom configuration (*xml2ucdb.ini*) file, use -noautonumber to subsequently turn it off.

- -datafields <str>

Specifies data fields, in the exact order that the columns appear in the testplan UCDB being imported. The -datafields switch uses a set of pre-defined keywords, which are treated by QuestaSim as a kind of “reserved” word. These case sensitive, order dependent, reserved words are: “Section”, “Title”, “Description”, “Link”, “Type”, “Path”, “Weight”, and “Goal”. Use these words, separated by the character specified with **-tagseparator**, to specify the fields in the XML test plan. This switch can also be used to add additional, user-

defined data fields to the testplan. Blank fields in XML source can be specified with the null character (“-”).

- **-datalabels <str>**
Specifies labels for datafields. Optional.
- **-datatags <tag>**
Specifies XML tag for item data fields. Optional.
- **-debug**
Prints out internal debug information. Optional.
- **-descriptiontag <tag>**
Specifies XML tag or tag list for description fields. Optional.
- **-dofilename <file>**
Specifies the name of test plan mapping file. The test plan mapping file contains the coverage tag commands necessary to tag and link the coverage objects to the test plan items. Optional. The default: no mapping file.
- **-excludetags <tags>**
Specifies XML tag or tag list for tags to exclude from the processing. Optional.
- **-excelsheet <sheet_name>**
Imports data from one specific sheet in an Excel spreadsheet. <sheet_name> is the exact string as it appears in the tab (“Sheet1”, “Sheet2”, etc.) at the bottom of an Excel spreadsheet. Optional.
- **-format <format>**
Specifies the format to be used for the command, from a list of formats (pre-defined XML semantic definitions) listed in the *xml2ucdb.ini* file, located in the <install_dir>/vm_src directory. Optional.
- **-formatlist**
Lists the currently defined formats (pre-defined XML semantic definitions in Tcl) listed in the *xml2ucdb.ini* file located in the <install_dir>/vm_src directory. Optional.
- **-goaltag <tags>**
Specifies the XML tag or tag list for a goal field. Optional.
- **-help**
Prints Help Message. Optional.
- **-inherit**
Used with a nested testplan. Causes storing state (resulting from start/stop tags or startstoring tag) to be inherited by the nested testplan.

- **-linkattr <tag>**
Specifies XML tag attribute for cover items. Optional.
- **-linktag <tag>**
Specifies XML tag or tag list for cover items. Optional.
- **-modelsimini <ini_filepath>**
Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).
- **-root <str>**
Specifies the root name to be prepended to each section number (in non-autonumbered testplans, i.e. spreadsheets) of the test plan. Allows you to specify a root test plan, in which you wish to “nest” the current test plan, thereby creating a hierarchical test plan. Optional.
- **-searchpath <path/to/XML_input>**
Specifies the path where XML import file is to be found. If this switch is not specified for a hierarchical (nested) test plan, any existing setting for parent test plan(s) is inherited. If none in ancestor test plans, the setting in the *xml2ucdb.ini* file is used. Optional.
- **-sectiontags <tag>**
Specifies XML tag or tag list for a test item section number (such as "tag1:tag2:tag3"). Optional.
- **-sectionprefix <str>**
Specifies the prefix for section numbering (such as "tag1prefix:tag2prefix:tag3prefix"). Optional.
- **-startsection <num>**
Sets starting item number: defaults 0. Optional.
- **-startstoring <num>**
Specifies the Storing to begin at an item number: defaults 0. Optional.
- **-starttags <tags>**
Specifies XML tag or tag list for a tag to start the processing. Optional.
- **-stoptags <tags>**
Specifies XML tag or tag list for a tag to stop the processing. Optional.
- **-stylesheet <file>**
Specifies the name of XSL pre-processing stylesheet to be used. Optional.

- **-tagprefix <str>**
Specifies a string to be prefixed to UCDB tag names. For a top-level testplan, if the tagprefix is not set, the value specified with **-title** is used. If any whitespace is contained in the title, it is replaced with underscore characters for use as the tagprefix. For nested testplans, there is no default tagprefix: if you do not specify a tagprefix, none is used for the nested testplan. Optional.
- **-tagseparators <str>**
Specifies a list of characters used as tag separators for tag arguments accepting multiple tags. Applies only to the taglist parameters (-starttags, -stoptags, excludetags, etc.) specified on the command line. Optional.
- **-title <str>**
Specifies a string to be used as the title for the test plan. For a top-level testplan, if the title is not set, the basename of the input XML file is used. For nested testplans, there is no default title: if you do not specify a title, none is used for the nested testplan. Optional.
- **-titletag <tag>**
Specifies XML tag or tag list for a test item name, or start tag for Data fields. Optional.
- **-typeattr <name>**
Specifies an attribute containing the "type" of each coverage item (coverpoint, covergroup, etc.). Types are case-insensitive. Optional.
- **<ucdb_filename>**
Specifies the name for the *.ucdb* output file. Required, unless -ucdbfilename is specified.
- **-ucdbfilename <file>**
Specifies the name for the *.ucdb* output file. Required, unless <ucdb_filename> is specified.
- **-verbose**
Prints the testplan hierarchy and design mapping. Optional.
- **-version**
Prints the version number of the utility. Optional.
- **-viewall**
Prints both tags and text contents of XML File. Optional.
- **-viewtags**
Prints tag contents of XML File. Optional.
- **-weighttag <tags>**
Specifies the XML tag or tag list for a weight field. Optional.

- `<XML_filename>`

Specifies the input XML file to be converted. Required. The path can be a full or relative path to the file location. On Windows systems the path separator should be a slash (/), rather than a backslash (\), for example:

```
C:/design/vplan/verification.xml
```

Examples

- Convert a user-customized XML file, specifying the required columns in the test plan, skipping a field (-), and adding a OWNER and RESPONSIBLE field:

```
xml2ucdb -datafields=Section,Title,Description,OWNER,  
-,RESPONSIBLE,Type,Weight,Goal
```

- Convert the an Excel formatted XML file called *input.xml* to a UCDB format file, *output.ucdb*:

```
xml2ucdb -format Excel input.xml output.ucdb
```

- Convert only a specified sheet, *Sheet1*, of an Excel formatted XML file called *input.xml* to a UCDB format file, *output.ucdb*:

```
xml2ucdb -format Excel -excelsheet Sheet1 input.xml output.ucdb
```

See also

[“xml2ucdb.ini Configuration File”](#), [coverage goal](#), [coverage tag](#), [coverage unlinked](#), [coverage weight](#), [vcover attribute](#), [vcover merge](#), [vcover testnames](#)

Chapter 3

AVM Encyclopedia

The AVM Encyclopedia documents all of the classes in the AVM library. The classes are organized in related groups. For each class there is a description of the class and what it is used for along with a listing of all the members and methods. The methods and members are described as well.

To use the Encyclopedia, look up a class name in the class index to find the page that has the complete description and the file name in the library where the class is defined. You can also peruse the class groups to understand how the classes work together.

Class Index

[Table 3-1](#) is a complete list of all the class definitions in alphabetic order, the file in which each definition resides, and a reference to a page number for the complete description of the class.

Table 3-1. Class Index

Class Name	Definition File	Page
analysis_imp	deprecated/tlm_imps.svh	826
analysis_port	deprecated/analysis_port.svh	827
avm_algorithmic_comparator	utils/avm_algorithmic_comparator.svh	793
avm_analysis_export	tlm/avm_exports.svh	800
avm_analysis_imp	tlm/avm_imps.svh	802
avm_analysis_port	tlm/avm_ports.svh	806
avm_blocking_get_export	tlm/avm_exports.svh	800
avm_blocking_get_imp	tlm/avm_imps.svh	802
avm_blocking_get_peek_export	tlm/avm_exports.svh	800
avm_blocking_get_peek_imp	tlm/avm_imps.svh	802
avm_blocking_get_peek_port	tlm/avm_ports.svh	804
avm_blocking_get_port	tlm/avm_ports.svh	804
avm_blocking_master_export	tlm/avm_exports.svh	800
avm_blocking_master_imp	tlm/avm_imps.svh	807

Table 3-1. Class Index

Class Name	Definition File	Page
avm_blocking_master_port	t1m/avm_ports.svh	804
avm_blocking_peek_export	t1m/avm_exports.svh	800
avm_blocking_peek_imp	t1m/avm_imps.svh	802
avm_blocking_peek_port	t1m/avm_ports.svh	804
avm_blocking_put_export	t1m/avm_exports.svh	800
avm_blocking_put_imp	t1m/avm_imps.svh	802
avm_blocking_put_port	t1m/avm_ports.svh	804
avm_blocking_slave_export	t1m/avm_exports.svh	800
avm_blocking_slave_imp	t1m/avm_imps.svh	809
avm_blocking_slave_port	t1m/avm_ports.svh	804
avm_built_in_clone	vbase/avm_policies.svh	864
avm_built_in_comp	vbase/avm_policies.svh	865
avm_built_in_converter	vbase/avm_policies.svh	866
avm_built_in_pair	utils/avm_pair.svh	867
avm_class_clone	vbase/avm_policies.svh	868
avm_class_comp	vbase/avm_policies.svh	869
avm_class_converter	vbase/avm_policies.svh	870
avm_class_pair	utils/avm_pair.svh	871
avm_connector_base	vbase/avm_connector_base.svh	811
avm_env	utils/avm_env.svh	779
avm_get_export	t1m/avm_exports.svh	800
avm_get_imp	t1m/avm_imps.svh	802
avm_get_peek_export	t1m/avm_exports.svh	800
avm_get_peek_imp	t1m/avm_imps.svh	802
avm_get_peek_port	t1m/avm_ports.svh	804
avm_get_port	t1m/avm_ports.svh	804
avm_in_order_built_in_comparator	utils/avm_in_order_comparator.svh	795
avm_in_order_class_comparator	utils/avm_in_order_comparator.svh	796
avm_in_order_comparator	utils/avm_in_order_comparator.svh	797

Table 3-1. Class Index

Class Name	Definition File	Page
avm_master_export	tlm/avm_exports.svh	800
avm_master_imp	tlm/avmimps.svh	815
avm_master_port	tlm/avm_ports.svh	804
avm_named_component	vbase/avm_named_component.svh	781
avm_nonblocking_get_export	tlm/avm_exports.svh	800
avm_nonblocking_get_imp	tlm/avmimps.svh	802
avm_nonblocking_get_peek_export	tlm/avm_exports.svh	800
avm_nonblocking_get_peek_imp	tlm/avmimps.svh	802
avm_nonblocking_get_peek_port	tlm/avm_ports.svh	804
avm_nonblocking_get_port	tlm/avm_ports.svh	804
avm_nonblocking_master_export	tlm/avm_exports.svh	800
avm_nonblocking_master_imp	tlm/avmimps.svh	817
avm_nonblocking_master_port	tlm/avm_ports.svh	804
avm_nonblocking_peek_export	tlm/avm_exports.svh	800
avm_nonblocking_peek_imp	tlm/avmimps.svh	802
avm_nonblocking_peek_port	tlm/avm_ports.svh	804
avm_nonblocking_put_export	tlm/avm_exports.svh	800
avm_nonblocking_put_imp	tlm/avmimps.svh	802
avm_nonblocking_put_port	tlm/avm_ports.svh	804
avm_nonblocking_slave_export	tlm/avm_exports.svh	800
avm_nonblocking_slave_imp	tlm/avmimps.svh	819
avm_nonblocking_slave_port	tlm/avm_ports.svh	804
avm_peek_export	tlm/avm_exports.svh	800
avm_peek_imp	tlm/avmimps.svh	802
avm_peek_port	tlm/avm_ports.svh	804
avm_port_base	vbase/avm_port_base.svh	821
avm_put_export	tlm/avm_exports.svh	800
avm_put_imp	tlm/avmimps.svh	802
analysis_fifo	tlm/tlm_fifos.svh	831

Table 3-1. Class Index

Class Name	Definition File	Page
analysis_if	tlm/tlm_ifs.svh	839
avm_put_port	tlm/avm_ports.svh	804
avm_random_stimulus	utils/avm_random_stimulus.svh	786
avm_report_client	reporting/avm_report_client.svh	874
avm_report_handler	reporting/avm_report_handler.svh	879
avm_report_server	reporting/avm_report_server.svh	882
avm_reporter	reporting/avm_report_client.svh	884
avm_slave_export	tlm/avm_exports.svh	800
avm_slave_imp	tlm/avm_imps.svh	823
avm_slave_port	tlm/avm_ports.svh	804
avm_stimulus	deprecated/avm_stimulus.svh	788
avm_subscriber	utils/avm_subscriber.svh	789
avm_threaded_component	utils/avm_threaded_component.svh	790
avm_transaction	vbase/avm_transaction.svh	872
avm_transport_export	tlm/avm_exports.svh	800
avm_transport_imp	tlm/avm_imps.svh	825
avm_transport_port	tlm/avm_ports.svh	804
avm_verification_component	deprecated/avm_verification_component.svh	791
global_analysis_ports	deprecated/avm_global_analysis_ports.svh	828
tlm_blocking_get_if	tlm/tlm_ifs.svh	800
tlm_blocking_get_imp	deprecated/tlm_imps.svh	829
tlm_blocking_get_peek_if	tlm/tlm_ifs.svh	800
tlm_blocking_get_peek_imp	deprecated/tlm_imps.svh	829
tlm_blocking_master_if	tlm/tlm_ifs.svh	800
tlm_blocking_master_imp	deprecated/tlm_imps.svh	829
tlm_blocking_peek_if	tlm/tlm_ifs.svh	800
tlm_blocking_peek_imp	deprecated/tlm_imps.svh	829
tlm_blocking_put_if	tlm/tlm_ifs.svh	800
tlm_blocking_put_imp	deprecated/tlm_imps.svh	829
tlm_blocking_slave_if	tlm/tlm_ifs.svh	800

Table 3-1. Class Index

Class Name	Definition File	Page
tlm_blocking_slave_imp	deprecated/tlm_imps.svh	829
tlm_fifo	tlm/tlm_fifos.svh	832
tlm_get_if	tlm/tlm_ifs.svh	846
tlm_get_imp	deprecated/tlm_imps.svh	829
tlm_get_peek_if	tlm/tlm_ifs.svh	847
tlm_get_peek_imp	deprecated/tlm_imps.svh	829
tlm_master_if	tlm/tlm_ifs.svh	849
tlm_master_imp	deprecated/tlm_imps.svh	829
tlm_nonblocking_get_if	tlm/tlm_ifs.svh	851
tlm_nonblocking_get_imp	deprecated/tlm_imps.svh	829
tlm_nonblocking_get_peek_if	tlm/tlm_ifs.svh	852
tlm_nonblocking_get_peek_imp	deprecated/tlm_imps.svh	829
tlm_nonblocking_master_if	tlm/tlm_ifs.svh	853
tlm_nonblocking_master_imp	deprecated/tlm_imps.svh	829
tlm_nonblocking_peek_if	tlm/tlm_ifs.svh	855
tlm_nonblocking_peek_imp	deprecated/tlm_imps.svh	829
tlm_nonblocking_put_if	tlm/tlm_ifs.svh	856
tlm_nonblocking_put_imp	deprecated/tlm_imps.svh	829
tlm_nonblocking_slave_if	tlm/tlm_ifs.svh	857
tlm_nonblocking_slave_imp	deprecated/tlm_imps.svh	829
tlm_peek_if	tlm/tlm_ifs.svh	859
tlm_peek_imp	deprecated/tlm_imps.svh	829
tlm_put_if	tlm/tlm_ifs.svh	860
tlm_put_imp	deprecated/tlm_imps.svh	829
tlm_req_rsp_channel	tlm/tlm_req_rsp.svh	835
tlm_slave_if	tlm/tlm_ifs.svh	861
tlm_slave_imp	deprecated/tlm_imps.svh	829
tlm_transport_channel	tlm/tlm_req_rsp.svh	838
tlm_transport_if	tlm/tlm_ifs.svh	863
tlm_transport_imp	deprecated/tlm_imps.svh	829

Classes for Components

Components form the foundation of the AVM. Components encapsulate behavior of transactors, scoreboards, and other objects in a test bench. `avm_named_component` is the base class from which all component classes are derived.

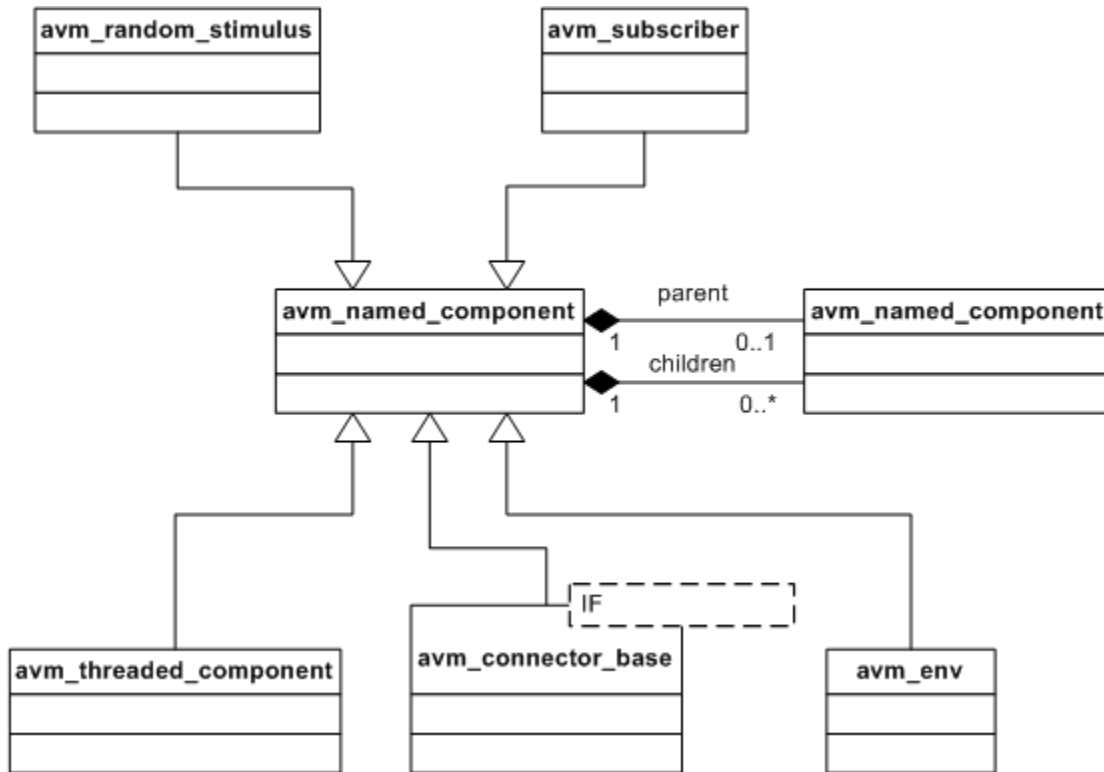


Figure 3-1. UML Diagram for Components

avm_env

extends `avm_named_component`

A subclass of `avm_env` is the top-level class in any class-based AVM verification environment. All the components of the test bench are children of this top-level class. There are two key methods in any such subclass: the constructor that specifies the connectivity between the environment class and the rest of the test bench and `do_test()`, which is inherited from the `avm_env` base class. `avm_env::do_test()` runs through the AVM phases. These are `connect()` (further broken down into `export_connections()`, `connect()` and `import_connections()`), `configure()`, `run()`, and `report()`.

file

`utils/avm_env.svh`

virtual

yes

members

`avm_connection_phase_e m_connection_phase =`
`AVM_CONSTRUCTION_PHASE`

Used to determine which phase of elaboration is currently being executed. This enum is used by `avm_connector_base` (see page 811) to check that the correct connections are being done in the correct subphase of the connect phase.

internal members

local `process m_run_process`
The run process is the process id of the `run()` task.

methods

function `new(string name="env")`
This is the constructor. `name` is the first argument of the normal `avm_named_component` constructor. It is not necessary to specify the parent argument, since `avm_env` does not have a parent.

virtual function void `configure()`
Allows configuration of verification components before the simulation starts, although it may be that back door memory accesses are also done here. It is virtual so that it can be overloaded in subclasses of `avm_env`. It is a function, so it cannot consume time. If there is some time-consuming initialization that needed before the test starts, then this needs to be done in the run phase. `configure()` is executed after `elaborate()` and before `run()`.

virtual function void `connect()`
A virtual method that gets overridden in the user-defined subclass to specify the connections between top-level components in the environment.

function void **do_kill_all()**

Kills all `run()` tasks, whether these were created by `avm_env` or `avm_threaded_component`. It is called by `do_test()` between the run phase and the report phase. It can also be called manually to stop all processes.

virtual task **do_test()**

The main user-visible method in `avm_env`. It runs through the AVM phases as described above.

virtual task **execute()**

This method is deprecated. It exists to support backward compatibility with AVM 2.0.

virtual function void **kill()**

Kills the `run()` task and any of its children. It can be overloaded to do additional work before or after killing these processes. However, it is necessary that the overloaded method call `super.kill()`.

virtual function void **report()**

Called when the `run()` task finishes. It provides a summary of all AVM message reporting calls, so if it is overridden in a subclass, it is recommended that `super.report()` be called as the last thing in the user-defined `report()` method. It is a function and cannot consume time. If some time-consuming post processing is needed, then it must be done at the end of the `run()` task.

virtual task **resume()**

Suspends the `run()` task and any of its children. It can be overloaded to do additional work before or after resuming these processes. However, it is necessary that the overloaded method call `super.resume()`.

virtual task **run()**

The main process at the top level of the test bench. Typically, it is used to start stimulus generation, examine the state of scoreboards and/or coverage objects, and then stop stimulus generation.

virtual task **suspend()**

Suspends the `run()` task and any of its children. It can be overloaded to do additional work before or after suspending these processes. However, it is necessary that the overloaded method call `super.suspend()`.

internal methods

task **do_run_all()**

A virtual method in `avm_named_component`. The implementation in `avm_env` adds a delta delay to ensure that all subcomponents start before the `run()` task in the `avm_env`.

local function void **elaborate()**

A local method that runs through the connect subphases. It is the first phase executed after the environment constructor completes.

avm_named_component

extends `avm_report_client`

This is the fundamental building block of the AVM. All structural classes (e.g., `avm_env`, `avm_threaded_component`, `avm_random_stimulus`, etc.) inherit from `avm_named_component`.

Broadly speaking, the methods of this class are divided into three kinds: the basic hierarchy handling methods, including the constructor and `remove`; the methods called by `avm_env` to do configuration, connections, and reporting; and the hierarchical reporting methods.

file

`vbase/avm_named_component.svh`

virtual

Yes

members

protected `avm_named_component` **`m_children[string]`**

An associative array of those named components that are children of this component. The children are indexed by their leaf name.

protected `avm_named_component` **`m_components[string]`**

An array used purely for debugging purposes. It is only valid after the end of elaboration. It consists of those children of this component that are not ports, exports, or implementations.

`avm_env` **`m_env`**

A handle to the `avm_env` within which this component is defined. It is `null` if the component is defined outside of an `avm_env`.

protected `avm_named_component` **`m_exports[string]`**

An array used purely for debugging purposes. It is only valid after the end of elaboration. It consists of those children of this component that are exports.

protected `avm_named_component` **`m_implementations[string]`**

An array used purely for debugging purposes, it is only valid after the end of elaboration. It consists of those children of this component that are implementations.

string **`m_leaf_name`**

The local name of the component, e.g., `data_phase_monitor`.

string **`m_name`**

The full hierarchical name of the component, e.g., `top.monitor.data_phase_monitor`.

protected `avm_named_component` **`m_parent`**

A handle to the parent. For `avm_envs` and components defined outside of `avm_envs` (e.g., in modules, interfaces, or program blocks) this will be `null`. For any component defined within an `avm_env`, the parent will be the `avm_env`.

protected `avm_named_component` **`m_ports[string]`**

An array used purely for debugging purposes. It is only valid after the end of elaboration. It consists of those children of this component that are ports.

internal members

local bit **m_is_removed**

Set to true when a component is removed.

static protected avm_env **s_current_env**

Only used for printing AVM 2.0 backward compatibility messages. It should not under any circumstances be relied on to contain a useful handle by subclasses of this component.

methods

function **new**(string name, avm_named_component parent=null,
bit check_parent=1)

We must always provide a local name. For most components, a parent is supplied and checked. The only exceptions to this are `avm_env`, which must not have a parent, and components such as `tlm_fifo` (see page 832), which might not have a parent if they are being used outside of an `avm_env`, such as in a module. If a component instantiation does not have a parent, then `check_parent` should be set to 0.

function avm_named_component **absolute_lookup**(string name)

Returns a handle to the component with the full hierarchical name specified by `name`, if there is such a component, and will return `null` otherwise.

virtual function void **configure**()

An empty implementation in `avm_named_component`. It should be overloaded if required in a subclass. It is called by the `avm_env` using `do_configure()`.

protected virtual function void **connect**()

Called by `avm_env` after `export_connections` and before `import_connections`. It should be overloaded in subclasses of `avm_named_component` so that a child port that requires an interface can obtain it from an export of another sibling child.

Connections in `connect()` should be of the form `child1.port.connect(child2.export)`.

virtual function void **do_display**(int max_level=-1,
int level=0,
bit display_connectors=0)

A debugging method that is used to recursively display the hierarchical names of this component and its children. `max_level` is used to control the depth of the recursion—the default value of -1 means that the recursion will always carry on to the lowest level in the hierarchy of the test bench. It is not expected that the normal test bench code will supply a value other than zero to the second argument. The third argument is used to control whether ports, exports, and implementations are displayed. The default value of zero indicates that they are not displayed; a value of 1 ensures that connectors are displayed.

virtual function void **do_kill_all()**

Kills all the `run()` tasks in the current instance of `avm_named_component` and any tasks spawned by this instance and any child component instances. It is called by `avm_env` after its `run()` task has finished executing.

function void **do_flush()**

Calls the virtual `flush()` method for this component and all its children using a bottom-up ordering. It is not called automatically by `avm_env`, so it needs to be called explicitly when required.

virtual function void **end_of_elaboration()**

A virtual function whose default implementation is empty. It is called by `avm_env` at the end of elaboration and before `configure()`. It can be overloaded in a subclass, and is a useful place to put debugging code that can display interesting aspects of the test bench hierarchy or connectivity.

protected virtual function void **export_connections()**

Called by `avm_env` at the beginning of the `connect()` phase. It should be overloaded in subclasses of `avm_named_component` to make `avm*_exports` and `avm*_imps` defined in children of this component externally visible. Connections in `export_connections` should be of the form `export.connect(child.export)`.

virtual function void **flush()**

An empty implementation in `avm_named_component`. It should be overloaded if required in a subclass. It is called from normal test bench code by `do_flush()`.

protected virtual function void **import_connections()**

Called by `avm_env` at the end of the `connect()` phase. It should be overloaded in subclasses of `avm_named_component` so that a child port that requires an external interface can obtain it from a port of this component. Connections in `import_connections` should be of the form `child.port.connect(port)`.

function bit **is_removed()**

Returns 1 if this component has been removed, otherwise returns 0.

function `avm_named_component` **relative_lookup(string name)**

Looks up the child of this component whose name relative to this component is `name`. For example, if this component's name is "i1.i2" and `name` is "i3.i4," then this method will return the handle to the component with name "i1.i2.i3.i4" if there is such a component, and will return `null` otherwise.

virtual function void **remove()**

Removes all trace of a component from the various AVM data structures. It is virtual to allow subclasses to delete this component from their data structures if that is necessary. `remove()` can only be called during the `avm_env` construction phase.

virtual function void **report()**

An empty implementation in `avm_named_component`. It should be overloaded if required in a subclass. It is called by the `avm_env` using `do_report()`.

function void **set_report_default_file_hier(FILE f)**

Calls `set_report_default_file()` on this component and all its children.

function void **do_export_connections()**
 Called by `avm_env` at the beginning of the `connect()` phase. Calls `export_connections()` in this component and all its children using a bottom-up ordering.

function void **do_import_connections()**
 Called by `avm_env` after `do_connect()`. Calls `import_connections()` in this component and all its children using a top-down ordering.

function void **do_report()**
 Called by `avm_env` after the `run()` method terminates. Calls `report` on this component and its children using a bottom-up ordering.

virtual task **do_run_all()**
 Spawns all the `run()` tasks in this component and all its children. Called by `avm_env` after the `configure()` phase and immediately before it spawns its own `run()` method.

function void **do_set_env(avm_env e)**
 Called by `avm_env` after construction and before the connection phase. It sets `m_env` in all the children of the `avm_env`.

local function void **extract_name()**
 A utility used by the absolute and relative look-up methods.

local function void **no_parent_message()**
 An internal method that prints out some AVM 2.0 to AVM 3.0 migration messages.

avm_random_stimulus

```
    #(type trans_type=avm_transaction)
    extends avm_named_component
```

This is a general purpose unidirectional random stimulus generator. It is a very useful component in its own right, but can also either be used as a template to define other stimulus generators, or it can be extended to add additional stimulus generation methods to simplify test writing.

The `avm_random_stimulus` class generates streams of `trans_type` transactions. These streams may be generated by the `randomize()` method of `trans_type`, or the `randomize()` method of one of its subclasses, depending on the type of the argument passed into the `generate_stimulus()` method. The stream may go indefinitely, until terminated by a call to `stop_stimulus_generation()`, or you may specify the maximum number of transactions to be generated.

By using inheritance, we can add directed initialization or tidy up sequences to the random stimulus generation.

file

```
utils/avm_random_stimulus.svh
```

virtual

```
no
```

parameters

```
type trans_type=avm_transaction
    Specifies the type of transaction to be generated.
```

members

```
avm_blocking_put_port #(trans_type) blocking_put_port
    The port through which transactions come out of the stimulus generator.

local bit m_stop=0
    Indicates whether the stimulus generator should stop before issuing the next
    transaction.
```

methods

```
function new(string name, avm_named_component parent)
    This is the standard AVM 3.0 constructor.
```

The constructor displays the string obtained from `get_randstate()` during construction. `set_randstate()` can then be used to regenerate precisely the same sequence of transactions for debugging purposes.

```
virtual task generate_stimulus(trans_type t=null,
                                int max_count=0)
```

The main user-visible method. If `t` is not specified, we will generate random transactions of type `trans_type`. If `t` is specified, we will use the `randomize()` method in `t` to generate transactions—so `t` must be a subclass of `trans_type`. `max_count` is the maximum number of transactions to be generated. A value of zero indicates no maximum—in this case, `generate_stimulus()` will go on indefinitely unless stopped by some other process. The transactions are cloned before they are sent out over the `blocking_put_port`.

virtual function void **stop_stimulus_generation()**
Stops the generation of stimulus.

avm_stimulus

```
    #(type trans_type=avm_transaction)
    extends avm_named_component
```

This is deprecated in AVM 3.0 in favor of `avm_random_stimulus`. It is in the library to ensure backward compatibility with AVM 2.0.

file

```
    deprecated/avm_stimulus.svh
```

virtual

```
    no
```

parameters

```
    type trans_type = avm_transaction
```

members

```
    tlm_blocking_put_if #(trans_type) blocking_put_port
    local bit m_stop=0
```

methods

```
    function new(string name, avm_named_component parent=null)
    virtual task generate_stimulus(trans_type t=null,
                                   int max_count=0)
    virtual function void stop_stimulus_generation()
```

avm_subscriber

```
 #(type T=int) extends avm_named_component
```

A subclass of `avm_subscriber` can be used to connect to an `avm_analysis_port` that writes transactions of type `T`. It has a single pure virtual method, `write()`, which is made available to the outside via an `analysis_export`. It is particularly useful when writing a coverage object that needs to be attached to a monitor.

file

```
utils/avm_subscriber.svh
```

virtual

```
yes
```

parameters

```
type T = int  
    Specifies the type of transaction to be received.
```

members

```
typedef avm_subscriber #(T) this_type  
avm_analysis_imp #(T, this_type) analysis_export  
    The export through which the write method is made available.
```

methods

```
function new(string name, avm_named_component p)  
    This is the standard AVM 3.0 constructor.  
  
pure virtual function void write(T t)  
    A pure virtual method that needs to be defined in a subclass.
```

avm_threaded_component

extends `avm_named_component`

A threaded component inherits from `avm_named_component` and adds the ability to spawn a `run()` task at the beginning of the simulation.

file

`utils/avm_threaded_component.svh`

virtual

yes

members

protected process `m_main_process`
The process id of the `run()` task.

methods

function `new(string name, avm_named_component parent)`
This is the standard AVM 3.0 constructor.

function void `do_kill_all()`
A virtual method in `avm_named_component` that is called by `avm_env` to kill all the `run()` tasks and any other processes spawned by any `run()` task.

virtual function void `kill()`
A virtual in `avm_named_component` that is called by `do_kill_all()`. If additional tidying up is required before or after killing the `run()` task and its children, then this method can be overloaded. If this is done, `super.kill()` must be called.

virtual function void `report()`
Called by `avm_env` after the run phase.

virtual task `resume()`
Resumes the `run()` task and any other processes spawned by the `run()` task.

pure virtual task `run()`
A pure virtual method that **must** be defined in any subclass.

virtual task `suspend()`
Suspends the `run()` task and any other processes spawned by the `run()` task.

internal methods

task `do_run_all()`
Used by the `avm_env` to spawn the `run()` method.

avm_verification_component

extends `avm_threaded_component`

This is deprecated in AVM 3.0 in favor of `avm_threaded_component`. It is in the library to ensure backward compatibility with AVM 2.0.

file

`deprecated/avm_verification_component`

virtual

yes

members

<none>

methods

```
function new(string name, avm_named_component parent)
```

Classes for Comparators

A common function of test benches is to compare streams of transactions for equivalence. For example, a test bench may compare a stream of transactions from a DUT with expected results. The AVM library provides a base class called `avm_in_order_comparator` and two derived classes, which are `avm_in_order_built_in_comparator` for comparing streams of built-in types and `avm_in_order_class_comparator` for comparing streams of class objects. `avm_algorithmic_comparator` also compares two streams of transactions; however, the transaction streams might be of different type objects. This device will use a user-written transformation function to convert one type to another before performing a comparison.

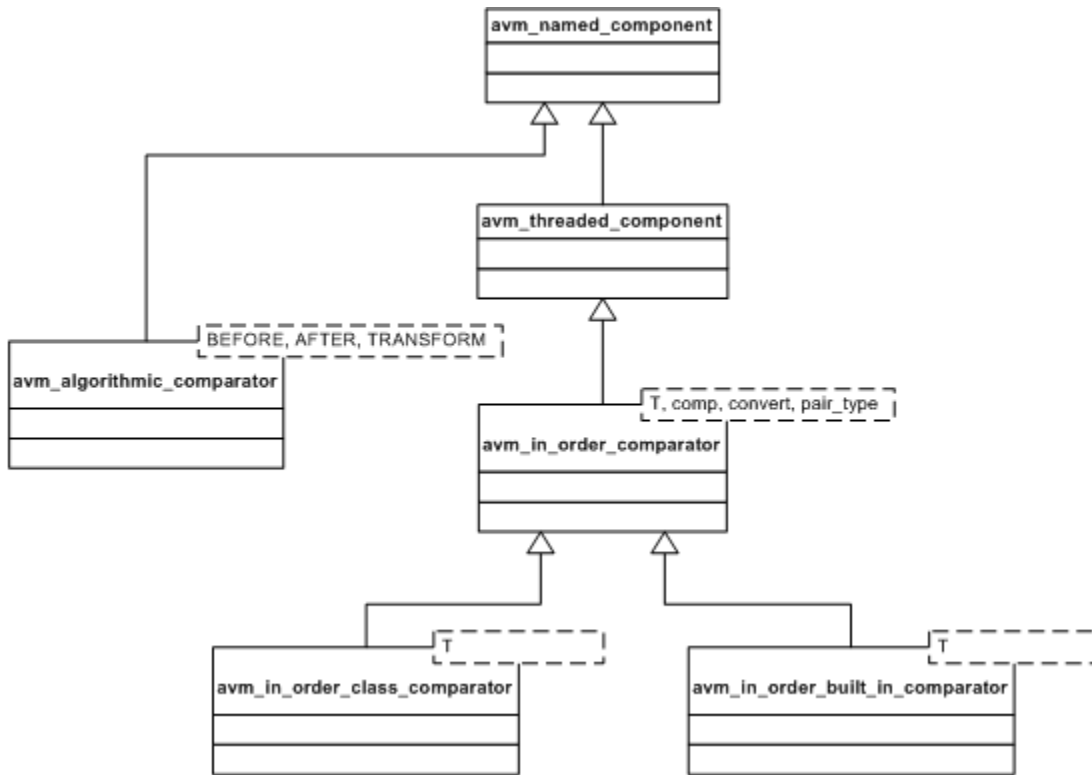


Figure 3-2. UML Diagram for Comparator Classes

avm_algorithmic_comparator

```

#(type BEFORE=int,
    type AFTER=int,
    type TRANSFORMER=int_transform)
extends avm_named_component

```

The algorithmic comparator is a wrapper around `avm_in_order_class_comparator`. Like the in-order comparator, the algorithmic comparator compares two streams of transactions, the “before” stream and the “after” stream. It is often the case when two streams of transactions need to be compared that the two streams are in different forms. That is, the type of the before transaction stream is different than the type of the after transaction stream.

The `avm_algorithmic_comparator` provides a transformer that transforms before transactions into after transactions. The transformer is supplied to the algorithmic comparator as a *policy class* via the class parameter `TRANSFORMER`. The transformer policy must provide a `transform()` method with the following prototype:

```
AFTER transform (BEFORE b);
```

file

```
utils/avm_algorithmic_comparator.svh
```

virtual

```
no
```

parameters

```
type AFTER = int
    The type of the transaction against which the transformed BEFORE transactions will
    be compared.
```

```
type BEFORE = int
    The type of incoming transaction to be transformed prior to comparing against the
    AFTER transactions.
```

```
type TRANSFORMER = int_transform
    The type of the class that contains the transform() method.
```

members

```
typedef avm_algorithmic_comparator
    #(BEFORE, AFTER, TRANSFORMER) this_type
```

```
avm_analysis_export #(AFTER) after_export
    Provides a write(AFTER t) method so that publishers (e.g., monitors) can send in an ordered
    stream of transactions against which the transformed BEFORE transactions will be compared.
```

```
avm_analysis_imp #(BEFORE, this_type) before_export
    Provides a write(BEFORE t) method so that publishers (e.g., monitors) can send in
    an ordered stream of transactions to be transformed and compared to the AFTER
    transactions.
```

internal members

local avm_in_order_class_comparator #(AFTER) **comp**
comp is the comparator used to compare the transformed BEFORE stream with the AFTER stream.

local TRANSFORMER **m_transformer**
m_transformer encapsulates the algorithm that transforms BEFORES into AFTERS.

methods

function **new**(TRANSFORMER transformer, string name,
avm_named_component parent)

The constructor takes a handle to an externally constructed transformer, a name, and a parent. The last two arguments are the normal arguments for an AVM 3.0 named component constructor.

We create an instance of the transformer (rather than making it a genuine policy class with a static transform method) because we might need to do reset and configuration on the transformer itself.

function void **export_connections**()

This is the standard AVM method for making exports and implementations of subcomponents visible externally.

function void **write**(BEFORE b)

This method handles incoming BEFORE transactions. It is usually accessed via the before_export, and it transforms the BEFORE transaction into an AFTER transaction before passing it to the in_order_class_comparator.

avm_in_order_built_in_comparator

```
 #(type T=int)  
 extends avm_in_order_comparator #(T)
```

A subclass of `avm_in_order_comparator` that is used to compare two streams of built-in types.

file

```
utils/avm_in_order_comparator.svh
```

virtual

```
no
```

parameters

```
type T = int  
    Specifies the type of transactions to be compared.
```

members

```
<none>
```

methods

```
function new(string name,avm_named_component parent)  
    This is the normal AVM 3.0 constructor
```

avm_in_order_class_comparator

```
 #(type T=int)
 extends avm_in_order_comparator #(T,
      avm_class_comp #(T),
      avm_class_converter #(T),
      avm_class_pair #(T))
```

A subclass of `avm_in_order_comparator` that is used to compare two streams of classes. It is assumed that the classes have `comp()` and `convert2string()` methods.

file

```
utils/avm_in_order_comparator.svh
```

virtual

```
no
```

parameters

```
type T = int
  Specifies the type of transactions to be compared.
```

members

```
<none>
```

methods

```
function new(string name, avm_named_component parent)
  This is the normal AMV 3.0 constructor
```

avm_in_order_comparator

```
 #(type T=int,  
     type comp=avm_built_in_comp #(T),  
     type convert=avm_built_in_converter #(T),  
     type pair_type=avm_built_in_pair #(T))  
 extends avm_threaded_component
```

Compares two streams of transactions. These transactions may either be classes or built-in types. To be successfully compared, the two streams of data must be in the same order. Apart from that, there are no assumptions made about the relative timing of the two streams of data.

file

utils/avm_in_order_comparator.svh

virtual

no

parameters

```
type T = int  
    Specifies the type of transactions to be compared.
```

```
type comp = avm_built_in_comp #(T)  
    The type of the comparator to be used to compare the two transaction streams.
```

```
type convert = avm_built_in_converter #(T)  
    A policy class to allow convert2string() to be called on the transactions being compared. If T is an extension of avm_transaction, it uses T::convert2string(). If T is a built-in type, the policy provides a convert2string() method for the comparator to call.
```

```
type pair_type = avm_built_in_pair #(T)  
    A policy class to allow pairs of transactions to be handled as a single avm_transaction type.
```

members

```
avm_analysis_export #(T) before_export  
    The export to which one stream of data is written.
```

```
avm_analysis_export #(T) after_export  
    The export to which the other stream of data is written.
```

```
avm_analysis_port #(pair_type) pair_ap  
    The comparator sends out pairs of transactions across this analysis port. Both matched and unmatched pairs are published.
```

```
int m_matches  
    The number of successfully paired transactions.
```

```
int m_mismatches  
    The number of unsuccessfully paired transactions.
```

members

local analysis_fifo #(T) **before_fifo**
The local storage for the stream of data coming in through before_export.

local analysis_fifo #(T) **after_fifo**
The local storage for the stream of data coming in through after_export.

methods

function **new**(string name, avm_named_component parent)
The normal AVM 3.0 constructor.

function void **export_connections**()
Connects the before_export and after_export to their respective FIFOs.

function void **flush**()
This method sets m_matches and m_mismatches back to zero. tlm_fifo::flush takes care of flushing the FIFOs.

task **run**()
Takes pairs of before and after transactions and compares them. Status information is updated according to the results of the comparison and pairs are published using the analysis port.

Classes for Connectors

Connectors are the ports and exports used to form transaction-level connections between components or between components and channels.

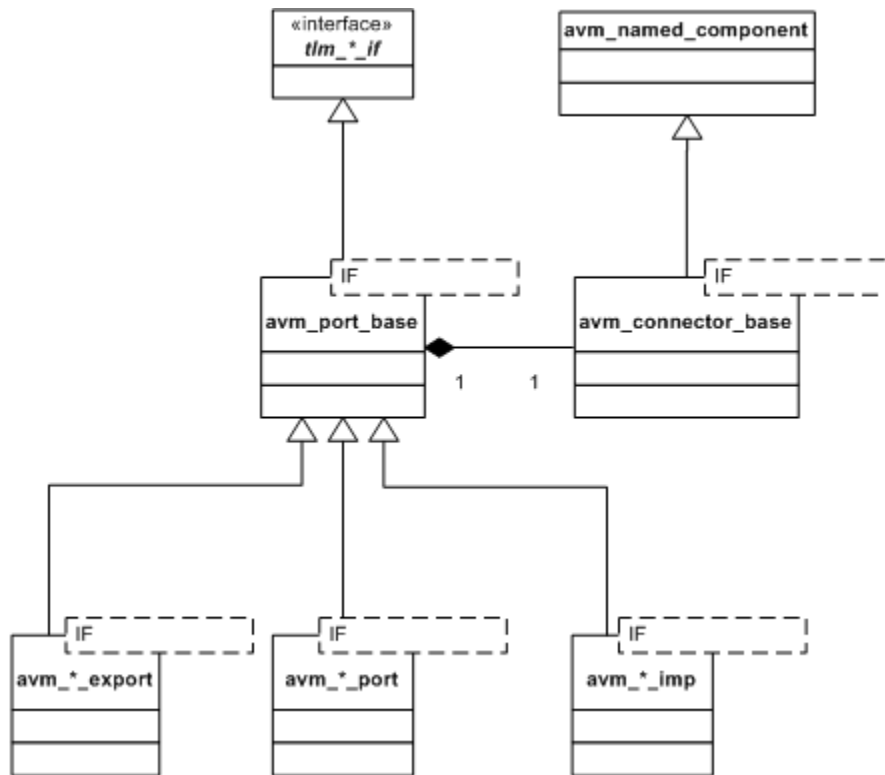


Figure 3-3. UML Diagram for Connectors

avm*_export

```
 #(type T=int) extends avm_port_base #(tlm*_if #(T))
```

An `avm*_export` is a connector that provides interfaces to other components. It gets these interfaces by connecting to an `avm*_export` or `avm*_imp` in a child component.

`avm*_export` inherits all the connectivity methods (e.g., the `connect()` method) from its base class `avm_port_base`.

It also implements all the methods of `tlm*_if`, as described in the documentation of `avm*_port` (see page 804). However, this is mainly for backwards compatibility with AVM 2.0. These methods are not usually called directly from normal test bench code.

file

```
tlm/avm_exports.svh
```

parameters

```
type T = int
```

The type of transaction to be communicated across the export.

members

```
<none>
```

methods

```
function new(string name, avm_named_component parent,  
             int min_size=1, int max_size=1)
```

`name` and `parent` are the standard AVM 3.0 constructor arguments. `min_size` and `max_size` specify the minimum and maximum number of interfaces that must have been supplied to this port by the end of elaboration.

The AVM library contains one export for each `tlm*_if` interface class as shown in [Table 3-2](#).

Table 3-2. Exports and Interfaces

Interface	Export
analysis_if	avm_analysis_export
tlm_blocking_get_if	avm_blocking_get_export
tlm_blocking_get_peek_if	avm_blocking_get_peek_export
tlm_blocking_master_if	avm_blocking_master_export
tlm_blocking_peek_if	avm_blocking_peek_export
tlm_blocking_put_if	avm_blocking_put_export
tlm_blocking_slave_if	avm_blocking_slave_export

Table 3-2. Exports and Interfaces

Interface	Export
tlm_get_if	avm_get_export
tlm_get_peek_if	avm_get_peek_export
tlm_master_if	avm_master_export
tlm_nonblocking_get_if	avm_nonblocking_get_export
tlm_nonblocking_get_peek_if	avm_nonblocking_get_peek_export
tlm_nonblocking_master_if	avm_nonblocking_master_export
tlm_nonblocking_peek_if	avm_nonblocking_peek_export
tlm_nonblocking_put_if	avm_nonblocking_put_export
tlm_nonblocking_slave_if	avm_nonblocking_slave_export
tlm_peek_if	avm_peek_export
tlm_put_if	avm_put_export
tlm_slave_if	avm_slave_export
tlm_transport_if	avm_transport_export

avm*_imp

```
 #(type T=int, type IMP=int)
 extends avm_port_base #(tlm*_if #(T))
```

avm*_imp provides a `tlm*_if` to ports and exports that require it. The actual implementation of the methods that comprise `tlm*_if` are defined in an object of type `IMP` (e.g., `tlm_fifo` `#(T)`) which is passed in to the constructor.

file

`tlm/avmimps.svh`

virtual

no

parameters

```
type T = int
    Type of transactions to be communicated across the underlying interface.
```

```
type IMP = int
    Type of the parent of this implementation.
```

internal members

```
local tlm*_if #(T) m_if
    Handle back to avm*_imp.
```

```
local IMP m_imp
    Handle to the component that implements the methods conveyed in the tlm*_if
    description.
```

methods

```
function new(string name, IMP imp)
```

`name` is the normal first argument to an AVM 3.0 constructor. `imp` is a slightly different form for the second argument to the AVM 3.0 constructor, which is of type `IMP` and defines the type of the parent.

Since it is the purpose of an “imp” class to provide an implementation of a set of interface tasks and functions, the particular set of tasks and functions available for each `avm*_imp` class is dependent on the type of the interface it implements, i.e., the particular TLM interface it extends.

[Table 3-3](#) lists all the `avm*_imp` classes and the interfaces each implements. The set of tasks and functions implemented is listed in the description of the interface classes.

Table 3-3. Interface Implementations

Implementation	Interface
avm_analysis_imp	analysis_if
avm_blocking_get_imp	tlm_blocking_get_if
avm_blocking_get_peek_imp	tlm_blocking_get_peek_if
avm_blocking_master_imp	tlm_blocking_master_if
avm_blocking_peek_imp	tlm_blocking_peek_if
avm_blocking_put_imp	tlm_blocking_put_if
avm_blocking_slave_imp	tlm_blocking_slave_if
avm_get_imp	tlm_get_if
avm_get_peek_imp	tlm_get_peek_if
avm_master_imp	tlm_master_if
avm_nonblocking_get_imp	tlm_nonblocking_get_if
avm_nonblocking_get_peek_imp	tlm_nonblocking_get_peek_if
avm_nonblocking_master_imp	tlm_nonblocking_master_if
avm_nonblocking_peek_imp	tlm_nonblocking_peek_if
avm_nonblocking_put_imp	tlm_nonblocking_put_if
avm_nonblocking_slave_imp	tlm_nonblocking_slave_if
avm_peek_imp	tlm_peek_if
avm_put_imp	tlm_put_if
avm_slave_imp	tlm_slave_if
avm_transport_imp	tlm_transport_if

avm*_port

```
 #(type T=int) extends avm_port_base #(tlm*_if #(T))
```

An `avm*_port` is a connector that requires interfaces to be supplied to it. It may get these interfaces by connecting to a parent's `avm*_port`, or an `avm*_export`, or `avm*_imp` in a sibling.

`avm*_port` inherits all the connectivity methods (e.g., the `connect()` method) from its base class, `avm_port_base`. It is effectively a proxy for the interface that originally supplied the implementation of this interface (e.g., a `tlm_fifo`). `avm*_port` implements all the methods in `tlm*_if`, and a call to any of these methods has the same effect as the equivalent call to the resolved implementation.

file

```
 tlm/avm_ports.svh
```

virtual

```
 no
```

parameters

```
 type T = int
     The type of transaction to be communicated across the port.
```

members

```
 <none>
```

methods

```
 function new(string name, avm_named_component parent,
              int min_size=1, int max_size=1)
     name and parent are the standard AVM 3.0 constructor arguments. min_size and
     max_size specify the minimum and maximum number of interfaces that must have
     been supplied to this port by the end of elaboration.
```

The set of functions and tasks available in each port object is dependent on the kind of port it is. [Table 3-4](#) lists, for each port type, the interface it implements. The tasks and functions for each interface can be found in the descriptions for the interface classes as shown below.

Table 3-4. Ports and Interfaces

Port	Interface
avm_analysis_port	analysis_if
avm_blocking_get_port	tlm_blocking_get_if
avm_blocking_get_peek_port	tlm_blocking_get_peek_if

Table 3-4. Ports and Interfaces

Port	Interface
avm_blocking_master_port	tlm_blocking_master_if
avm_blocking_peek_port	tlm_blocking_peek_if
avm_blocking_put_port	tlm_blocking_put_if
avm_blocking_slave_port	tlm_blocking_slave_if
avm_get_port	tlm_get_if
avm_get_peek_port	tlm_get_peek_if
avm_master_port	tlm_master_if
avm_nonblocking_get_port	tlm_nonblocking_get_if
avm_nonblocking_get_peek_port	tlm_nonblocking_get_peek_if
avm_nonblocking_master_port	tlm_nonblocking_master_if
avm_nonblocking_peek_port	tlm_nonblocking_peek_if
avm_nonblocking_put_port	tlm_nonblocking_put_if
avm_nonblocking_slave_port	tlm_nonblocking_slave_if
avm_peek_port	tlm_peek_if
avm_put_port	tlm_put_if
avm_slave_port	tlm_slave_if
avm_transport_port	tlm_transport_if

avm_analysis_port

```
    #(type T=int)
    extends avm_port_base #(analysis_if #(T))
```

`avm_analysis_port` is used by a component such as a monitor to publish a transaction to zero, one, or more subscribers. Typically, it will be used inside a monitor to publish a transaction observed on a bus to scoreboards and coverage objects.

file

```
tlm/avm_ports.svh
```

parameters

```
type T = int
```

The type of transaction to be written by the analysis port.

members

```
typedef avm_port_base #(analysis_if #(T)) port_type
```

methods

```
function new(string name, avm_named_component parent)
```

This is the standard AVM 3.0 constructor. `parent` should be `null` for analysis ports defined in a static scope, e.g., in a module-based monitor.

```
virtual function void connect(port_type provider)
```

Used to connect an analysis port to another analysis port, an analysis export, or an analysis implementation; e.g., in a flat hierarchy, we will typically use `monitor.ap.connect(coverage_object.analysis_export)` to connect a monitor to a coverage object observing the transactions being emitted by the monitor.

```
function void register(analysis_if #(T) _if)
```

Provides backwards compatibility with AVM 2.0.

```
function void write(T t)
```

Publishes transaction `t` to all subscribers.

avm_blocking_master_imp

```
 #(type REQ=int, type RSP=int,  
     type IMP=int,  
     type REQ_IMP=IMP, type RSP_IMP=IMP)  
 extends avm_port_base #(tlm_blocking_master_if #(REQ, RSP))
```

A blocking master implementation allows a single or a pair of components that implement `put(request)`, `get(response)`, and `peek(response)` to export a single interface that allows a master to put requests and get or peek responses.

file

tlm/avm_imps.svh

virtual

no

parameters

```
type REQ = int  
    Type of transactions to be sent out by this master.  
  
type RSP = int  
    Type of transactions to be received by this master.  
  
type IMP = int  
    Type of the parent of this implementation.  
  
type REQ_IMP = IMP  
    Type of the object that implements the request side of the interface.  
  
type RSP_IMP = IMP  
    Type of the object that implements the response side of the interface.
```

internal members

```
local tlm_blocking_master_if #(REQ, RSP) m_if  
    Handle back to the blocking master implementation itself.  
  
local REQ_IMP m_req_imp  
    Handle to the object that implements put(request), try_put(request) and can_put. By default, it is the parent of the nonblocking master implementation.  
  
local RSP_IMP m_rsp_imp  
    Handle to the object that implements get(response), try_get(request), can_get, peek(response), try_peek(response), and can_peek. By default, it is the parent of the nonblocking master implementation.
```

methods

```
function new(string name, IMP imp,  
            REQ_IMP req_imp=imp, RSP_IMP rsp_imp=imp)
```

`name` is the normal first argument to an AVM 3.0 constructor. `imp` is a slightly different form for the second argument to the AVM 3.0 constructor, which is of type `IMP` and defines the type of the parent. `req_imp` and `rsp_imp` are optional. If they are specified, then they must point to the underlying implementation of the request and response methods; e.g., in `t1m_req_rsp_channel` (see page [835](#)), `req_imp` and `rsp_imp` are the request and response FIFOs.

```
task put(input REQ req)  
task get(output RSP rsp)  
task peek(output RSP rsp)
```

See the documentation for `t1m_blocking_master_if` (see page [844](#)) for a description of these methods.

avm_blocking_slave_imp

```
 #(type REQ=int, type RSP=int, type IMP=int,  
    type REQ_IMP=IMP, type RSP_IMP=IMP)  
 extends avm_port_base #(tlm_blocking_slave_if #(REQ, RSP))
```

A blocking slave implementation allows a single or a pair of components that implement `put(response)`, `get(request)`, and `peek(request)` to export a single interface that allows a slave to get or peek requests and put responses.

file

tlm/avm_imps.svh

virtual

no

parameters

```
type REQ = int  
    Type of transactions to be received by this slave.  
  
type RSP = int  
    Type of transactions to be sent out by this master.  
  
type IMP = int  
    Type of the parent of this implementation.  
  
type REQ_IMP = IMP  
    Type of the object that implements the request side of the interface.  
  
type RSP_IMP = IMP  
    Type of the object that implements the response side of the interface.
```

internal members

```
local tlm_blocking_slave_if #(REQ, RSP) m_if  
    Handle back to the nonblocking slave implementation itself.  
  
local REQ_IMP m_req_imp  
    Handle to the object that implements get(request), can_get, peek(response),  
    and can_peek. By default, it is the parent of the blocking master implementation.  
  
local RSP_IMP m_rsp_imp  
    Handle to the object that implements put(response) and can_put. By default, it is  
    the parent of the blocking master implementation.
```

methods

```
function new(string name, IMP imp,  
            REQ_IMP req_imp=imp, RSP_IMP rsp_imp=imp)
```

`name` is the normal first argument to an AVM 3.0 constructor. `imp` is a slightly different form for the second argument to the AVM 3.0 constructor, which is of type `IMP` and defines the type of the parent. `req_imp` and `rsp_imp` are optional. If they are specified, then they must point to the underlying implementation of the request and response methods; e.g., in `t1m_req_rsp_channel` (see page [835](#)), `req_imp` and `rsp_imp` are the request and response FIFOs.

```
task put(input RSP rsp)
task get(output REQ req)
task peek(output REQ req)
```

See the documentation for `t1m_blocking_slave_if` (see page [845](#)) for a description of these methods.

avm_connector_base

```

#(type IF=int)
extends avm_named_component

```

avm_connector_base does all the work for ports, exports and implementations. A port, export, or implementation has a handle to a corresponding avm_connector_base and delegates most of the hard work to it. avm_connector_base actually does the connection between one connector and another (including all the checking), and it provides the implementation of the debugging methods. Because it is an avm_named_component (unlike the port, export, and imp classes that delegate to it) it appears in the AVM data structures and implements various utility methods that are virtual in avm_named_component.

file

```
vbase/avm_connector_base.svh
```

enums

```

typedef enum {AVM_PORT, AVM_EXPORT, AVM_IMPLEMENTATION}
             avm_port_type_e

```

Lists the types of connectors allowed in the AVM.

```

typedef enum {AVM_CONSTRUCTION_PHASE,
             AVM_EXPORT_CONNECTIONS_PHASE,
             AVM_CONNECT_PHASE,
             AVM_IMPORT_CONNECTIONS_PHASE,
             AVM_DONE_CONNECTIONS_PHASE}
             avm_connection_phase_e

```

Lists the phases executed during the elaboration of an avm_env.

virtual

```
no
```

parameters

```
type IF = int
```

A placeholder for the type of interface being required or provided by this connector.

internal members

```
typedef avm_connector_base #(IF) connector_type
```

```
local IF m_if_list[$]
```

Holds the interfaces that (should) satisfy the connectivity requirements of this connector. At the end of elaboration, an error will be reported if the size of this list is not between m_min_size and m_max_size (inclusive).

```
local int m_max_size
```

The maximum number of interfaces that this connector can have at the end of elaboration. This value is checked during elaboration.

```
local int m_min_size
```

The minimum number of interfaces that this connector can have at the end of elaboration. This value is checked at the end of elaboration.

- local avm_port_type_e **m_port_type**
Indicates whether this connector is a port, export, or implementation.
- local avm_connector_base #(IF) **m_provided_by[string]**
An associative array of connector bases that have supplied their interfaces to satisfy the connectivity requirements of this `avm_connector_base`. It is indexed by the name of the connector to make debugging easier. All the interfaces of the `avm_connector_bases` in `this.m_provided_by` are copied into `this.m_if_list`.
- local avm_connector_base #(IF) **m_provided_to[string]**
An associative array of `avm_connector_bases` to which this connector base has supplied its interfaces. It is indexed by the name of the connector to make debugging easier. All the interfaces of the `avm_connector_bases` in `this.m_if_list` are copied into `this.m_provided_to`.

methods

- function **new(string name, avm_named_component parent, avm_port_type_e port_type, int min_size, int max_size, bit check_parent)**
This is the constructor. The first two arguments are the standard AVM 3.0 constructor arguments. The `port_type` tells us whether this `connector_base` is a port, an export, or an implementation. `min_size` and `max_size` are the minimum and maximum number of interfaces that must be present at the end of elaboration. `check_parent` is also a standard AVM 3.0 constructor argument, which is used to indicate whether hierarchy checking should happen. For connectors, it is typically only used for `analysis_ports` in modules, which have no parent.
- function bit **add_if(IF _if)**
Used to connect AVM 3.0 connectors to AVM 2.0 exports and implementations. It simply copies in `_if` to this connector's interface list without updating the `provided_by` and `provided_to` lists.
- function void **check_connection_size()**
Checks the minimum connection size. The maximum connection size is checked as interfaces are added.
- local virtual function void **add_to_debug_list()**
Puts this `avm_connector_base`'s handle into the relevant associative array in its parent. This method is purely for debugging purposes.
- function void **check_min_connection_size()**
Checks that the minimum size for this connector has not been violated. It is called when an export or implementation supplies its interfaces to a port or export, and it is also called on every connector at the end of elaboration.
- function bit **check_phase(connector_type provider)**

Checks that legal connections are being made in the correct phase. Port->port connections must be done in `import_connections()`, port->export and port->imp must be done in `connect()`, and export->export and export->imp must be done in `export_connections()`. Anything else is an error.

function bit **check_relationship(connector_type provider)**

Checks that legal connections in the correct phase do not violate the required parent-child relationships. Connections made in `import_connections` must be of the type `child.port.connect(port)`; those made in `connect` must be of the form `child.port.connect(child.export)`; and those done in `export_connections` must be of the form `export.connect(child.export)`;

function bit **check_types(connector_type provider)**

Checks that only legal types are being connected. These are port -> port, port->export, export->export, and export->imp. Exports cannot connect to ports, andimps cannot connect to anything.

function bit **connect_to(input connector_type c)**

Copies the interfaces provided by `c` into this connector base. It also checks the minimum connection size of `c` and updates this connector's `provided_by` list and `c`'s `provided_to` list.

function void **debug_connected_to(int level=0, int max_level=-1)**

A debugging method that looks "forward" at the connections made that satisfy this connector's requirements. It is recursive to the depth specified by `max_level`. The default value of -1 means that the recursion continues until it hits an implementation, at which point it can follow the `provided_by` connections no further. It can be useful to call this method in `avm_named_component::end_of_elaboration`.

function void **debug_provided_to(int level=0, int max_level=-1)**

A debugging method that looks "backwards" at the connectors whose requirements are satisfied by this connector. It is recursive to the depth specified by `max_level`. The default value of -1 means that the recursion continues until it hits a leaf level port, at which point it can follow the `provided_to` connections no further. It can be useful to call this method in `avm_named_component::end_of_elaboration`.

function void **do_display(int max_level=-1, int level=0,**
 bit display_connectors=0)

A recursive virtual method originally defined in `avm_named_component`. It prints a high verbosity message if `display_connectors` is true. Otherwise, it does nothing.

function IF **lookup_indexed_if(int i=0)**

Looks up the i^{th} interface supplied to this connector. It is typically used to access the various interfaces bound to a multiport.

function int **max_size()**

Returns the maximum number of connected interfaces.

function int **min_size()**

Returns the minimum number of connected interfaces.

function int **size()**

Returns the number of connected interfaces (i.e., the number of elements in the `m_if_list`).

```
function void update_connection_lists(input connector_type c)  
    Updates this connector's provided_by list and c's provided_to list.
```

internal methods

```
local function avm_connection_phase_e  
    get_required_phase(avm_port_type_e provider_port_type)  
    Gets the current phase of the elaborator in the avm_env within which this connector  
    is defined.
```

avm_master_imp

```
 #(type REQ=int, type RSP=int, type IMP=int,  
     type REQ_IMP=IMP, type RSP_IMP=IMP)  
 extends avm_port_base #(tlm_master_if #(REQ, RSP))
```

A master implementation allows a single or pair of components that implement `put(request)`, `try_put(request)`, `can_put`, `get(response)`, `try_get(response)`, `can_get`, `peek(response)`, `try_peek(request)`, and `can_peek` to export a single interface that allows a master to put requests and get or peek responses in both blocking and nonblocking flavors.

file

tlm/avm_imps.svh

virtual

no

parameters

```
type REQ = int  
    Type of transactions to be sent out by this master.  
  
type RSP = int  
    Type of transactions to be received by this master.  
  
type IMP = int  
    Type of the parent of this implementation.  
  
type REQ_IMP = IMP  
    Type of the object that implements the request side of the interface.  
  
type RSP_IMP = IMP  
    Type of the object that implements the response side of the interface.
```

internal members

```
local tlm_master_if #(REQ, RSP) m_if  
    Handle back to the master implementation itself.  
  
local REQ_IMP m_req_imp  
    Handle to the object that implements put(request), try_put(request), and  
    can_put. By default, it is the parent of the master implementation.  
  
local RSP_IMP m_rsp_imp  
    Handle to the object that implements get(response), try_get(request),  
    can_get, peek(response), try_peek(response), and can_peek. By default, it is  
    the parent of the master implementation.
```

methods

```
function new(string name, IMP imp,  
            REQ_IMP req_imp=imp, RSP_IMP rsp_imp=imp)
```

`name` is the normal first argument to an AVM 3.0 constructor. `imp` is a slightly different form for a second argument to the AVM 3.0 constructor, which is of type `IMP` and defines the type of the parent. `req_imp` and `rsp_imp` are optional. If they are specified, then they must point to the underlying implementation of the request and response methods; e.g., in `tlm_req_rsp_channel` (see page [835](#)), `req_imp` and `rsp_imp` are the request and response FIFOs.

```
task put(input REQ req)
function bit try_put(input REQ req)
function bit can_put()
task get(output RSP rsp)
function bit try_get(output RSP rsp)
function bit can_get()
task peek(output RSP rsp)
function bit try_peek(output RSP rsp)
function bit can_peek()
```

See the documentation of `tlm_master_if` (see page [849](#)) for a description of these methods.

avm_nonblocking_master_imp

```

#(type REQ=int, type RSP=int,
    type IMP=int,
    type REQ_IMP=IMP,
    type RSP_IMP=IMP)
extends avm_port_base #(tlm_nonblocking_master_if #(REQ, RSP))

```

A nonblocking master implementation allows a single or pair of components that implement `try_put(request)`, `can_put`, `try_get(response)`, `can_get`, `try_peek(response)`, and `can_peek` to export a single interface that allows a master to put requests and get or peek responses.

file

tlm/avm_imps.svh

virtual

no

parameters

```

type REQ = int
    Type of transactions to be sent out by this master.

type RSP = int
    Type of transactions to be received by this master.

type IMP = int
    Type of the parent of this implementation.

type REQ_IMP = IMP
    Type of the object that implements the request side of the interface.

type RSP_IMP = IMP
    Type of the object that implements the response side of the interface.

```

internal members

```

local tlm_nonblocking_master_if #(REQ, RSP) m_if
    Handle back to the nonblocking master implementation itself.

local REQ_IMP m_req_imp
    Handle to the object that implements try_put(request) and can_put. By default, it is the parent of the nonblocking master implementation.

local RSP_IMP m_rsp_imp
    Handle to the object that implements try_get(response), can_get, try_peek(response), and can_peek. By default, it is the parent of the nonblocking master implementation.

```

methods

```

function new(string name, IMP imp,

```

`REQ_IMP req_imp=imp, RSP_IMP rsp_imp=imp`
`name` is the normal first argument to an AVM 3.0 constructor. `imp` is a slightly different form for the second argument to the AVM 3.0 constructor, which is of type `IMP` and defines the type of the parent. `req_imp` and `rsp_imp` are optional. If they are specified, then they must point to the underlying implementation of the request and response methods; e.g., in `t1m_req_rsp_channel` (see page [835](#)), `req_imp` and `rsp_imp` are the request and response FIFOs.

```
function bit try_put(input REQ req)
function bit can_put()
function bit try_get(output RSP rsp)
function bit can_get()
function bit try_peek(output RSP rsp)
function bit can_peek()
```

See the documentation for `t1m_nonblocking_master_if` (see page [853](#)) for a description of these methods.

avm_nonblocking_slave_imp

```
 #(type REQ=int, type RSP=int,  
                                     type IMP=int,  
                                     type REQ_IMP=IMP,  
                                     type RSP_IMP=IMP)  
 extends avm_port_base #(tlm_nonblocking_slave_if #(REQ, RSP))
```

A nonblocking slave implementation allows a single or pair of components that implement `try_put(response)`, `can_put`, `try_get(request)`, `can_get`, `try_peek(request)`, and `can_peek` to export a single interface that allows a slave to get or peek requests and put responses.

file

tlm/avm_imps.svh

virtual

no

parameters

```
type REQ = int  
    Type of transactions to be received by this slave.  
  
type RSP = int  
    Type of transactions to be sent out by this master.  
  
type IMP = int  
    Type of the parent of this implementation.  
  
type REQ_IMP = IMP  
    Type of the object that implements the request side of the interface.  
  
type RSP_IMP = IMP  
    Type of the object that implements the response side of the interface.
```

internal members

```
local tlm_nonblocking_slave_if #(REQ, RSP) m_if  
    Handle back to the nonblocking slave implementation itself.  
  
local REQ_IMP m_req_imp  
    Handle to the object that implements try_get(request), can_get,  
    try_peek(request), and can_peek. By default, it is the parent of the nonblocking  
    slave implementation.  
  
local RSP_IMP m_rsp_imp  
    Handle to the object that implements try_put(response) and can_put. By default,  
    it is the parent of the nonblocking slave implementation.
```

methods

```
function new(string name, IMP imp,
```

`REQ_IMP req_imp=imp, RSP_IMP rsp_imp=imp`
`name` is the normal first argument to an AVM 3.0 constructor. `imp` is a slightly different form for the second argument to the AVM 3.0 constructor, which is of type `IMP` and defines the type of the parent. `req_imp` and `rsp_imp` are optional. If they are specified, then they must point to the underlying implementation of the request and response methods; e.g., in `t1m_req_rsp_channel` (see page [835](#)), `req_imp` and `rsp_imp` are the request and response FIFOs.

```
function bit try_put(input RSP rsp)
function bit can_put()
function bit try_get(output REQ req)
function bit can_get()
function bit try_peek(output REQ req)
function bit can_peek()
```

See the documentation for `t1m_nonblocking_slave_if` (see page [857](#)) for a description of these methods.

avm_port_base

`#(type IF=avm_virtual_class) extends IF`
avm_port_base is the base class for all ports, exports, and implementations (avm*_port, avm*_export, and avm*_imp). avm_port_base extends IF, which is the type of the interface required and/or provided by the port, export, or implementation.

In many senses, avm_port_base is a facade class. It has a handle to an avm_connector_base and delegates much of the functionality to it.

file

vbase/avm_port_base.svh

virtual

yes

parameters

type **IF** = avm_virtual_class

A placeholder for the type of interface supported by this connector. The default value, avm_virtual_class, is a virtual class defined in vbase/avm_vbase.svh. Because it is virtual, a specific interface class (see “TLM Interfaces” on page 838) must be provided when extending the avm_port_base.

members

```
typedef avm_connector_base #(IF) connector_type
typedef avm_port_base #(IF) this_type
```

avm_connector_base #(IF) **m_connector**

The place where most of the hard work related to checking the validity of the connection, making the connection, and providing debugging information is done. Many of the methods below are delegated to m_connector.

protected IF **m_if**

A handle to the 0th interface that has been connected to this port, export, or implementation.

methods

```
function new(string name, avm_named_component parent,
             avm_port_type_e port_type,
             int min_size=1, int max_size=1,
             bit check_parent=1)
```

The first two arguments are the normal AVM 3.0 constructor arguments. The port_type is port, export, or implementation. min_size and max_size specify the minimum and maximum number of interfaces that must be supplied to this port base by the end of elaboration. parent is usually non null, in which case, check_parent should take its default value of 1. The rare exception to this (usually, analysis ports

defined outside of an `avm_env`) should set the value of `parent` to `null` and `check_parent` to 0.

function void **connect(this_type provider)**

Connects a port or export that requires interfaces of type `IF` to a port, export, or implementation that provides interfaces of type `IF`.

function void **connect_to_if(IF _if)**

Connects directly to an interface by delegating the call to `m_connector`. The main use for this method is to enable backward compatibility with AVM 2.0.

function void **debug_connected_to(int level=0, int max_level=-1)**

Prints out information on the connectors that have supplied interfaces to this connector, by delegating the call to `m_connector`.

function void **debug_provided_to(int level=0, int max_level=-1)**

Prints out information on the connectors that this connector has supplied interfaces to, by delegating the call to `m_connector`.

function IF **lookup_indexed_if(int i=0)**

Gets the i^{th} interface that has been provided to this port base, by delegating the call to `m_connector`.

function void **remove()**

Delegates the method call to `m_connector`.

function int **size()**

Gets the number of interfaces that have been provided to this port base by delegating the call to `m_connector`.

avm_slave_imp

```
 #(type REQ=int, type RSP=int, type IMP=int,  
    type REQ_IMP=IMP, type RSP_IMP=IMP)  
 extends avm_port_base #(tlm_slave_if #(REQ, RSP))
```

A slave implementation allows a single or pair of components that implement `put(response)`, `try_put(response)`, `can_put`, `get(request)`, `try_get(request)`, `can_get`, `peek(request)`, `try_peek(request)`, and `can_peek` to export a single interface that allows a slave to get or peek requests and put responses.

file

tlm/avm_imps.svh

virtual

no

parameters

```
type REQ = int  
    Type of transactions to be received by this slave.  
  
type RSP = int  
    Type of transactions to be sent out by this master.  
  
type IMP = int  
    Type of the parent of this implementation.  
  
type REQ_IMP = IMP  
    Type of the object that implements the request side of the interface.  
  
type RSP_IMP = IMP  
    Type of the object that implements the response side of the interface.
```

internal members

```
local tlm_slave_if #(REQ, RSP) m_if  
    Handle back to the slave implementation itself.  
  
local REQ_IMP m_req_imp  
    Handle to the object that implements get(request), try_get(request), can_get,  
    peek(request), try_peek(request), and can_peek. By default, it is the parent of  
    the slave implementation.  
  
local RSP_IMP m_rsp_imp  
    Handle to the object that implements put(response), try_put(response), and  
    can_put. By default, it is the parent of the slave implementation.
```

methods

```
function new(string name, IMP imp,  
            REQ_IMP req_imp=imp, RSP_IMP rsp_imp=imp)
```

`name` is the normal first argument to an AVM 3.0 constructor. `imp` is a slightly different form for the second argument to the AVM 3.0 constructor, which is of type `IMP` and defines the type of the parent. `req_imp` and `rsp_imp` are optional. If they are specified, then they must point to the underlying implementation of the request and response methods; e.g., in `t1m_req_rsp_channel` (see page [835](#)). `req_imp` and `rsp_imp` are the request and response FIFOs.

```
task put(input RSP rsp)
function bit try_put(input RSP rsp)
function bit can_put()
task get(output REQ req)
function bit try_get(output REQ req)
function bit can_get()
task peek(output REQ req)
function bit try_peek(output REQ req)
function bit can_peek()
```

See the documentation for `t1m_slave_if` (see page [861](#)) for a description of these methods.

avm_transport_imp

```
 #(type REQ=int, type RSP=int, type IMP=int)
   extends avm_port_base #(tlm_transport_if #(REQ, RSP))
```

A transport implementation allows a component that implements the transport task to export a `tlm_transport_if`.

file

```
tlm/avmimps.svh
```

virtual

```
no
```

parameters

```
type REQ = int
    Type of transactions to be received by this slave.
```

```
type RSP = int
    Type of transactions to be sent out by this master.
```

```
type IMP = int
    Type of the parent of this implementation.
```

internal members

```
local tlm_transport_if #(REQ, RSP) m_if
    Handle back to the avm_transport_imp itself.
```

```
local IMP m_imp
    Handle to the component that implements the transport task.
```

methods

```
function new(string name, IMP imp)
    name is the normal first argument to an AVM 3.0 constructor. imp is a slightly
    different form for the second argument to the AVM 3.0 constructor, which is of type
    IMP and defines the type of the parent.
```

```
task transport(input REQ request, output RSP response)
    Delegates the call to m_imp.transport().
```

analysis_imp

```
    #(type IMP=virtual_class, type T=int) extends analysis_if #(T)  
    [Deprecated in AVM-3.0. Use avm_analysis_imp instead.]
```

file

```
    deprecated/tlmimps.svh
```

virtual

```
    no
```

members

```
    local IMP m_imp  
    function new(IMP i)
```

methods

```
    function void write(input T t)
```

analysis_port

```

    #(type T=int) extends analysis_if #(T)
    [Deprecated in AVM-3.0. Use avm_analysis_port instead.]

```

file

```

    deprecated/analysis_port.svh

```

virtual

```

    no

```

parameters

```

    type T = int

```

members

```

    local analysis_if #(T) if_list[$]
    local avm_reporter r

```

methods

```

    function new()
    function void register(input analysis_if #(T) i)
    function void write(input T t)

```

global_analysis_ports

`#(type T=int)`

This is deprecated in AVM 3.0. Instead, use a normal `avm_analysis_port` and the absolute and relative look-up methods in `avm_named_component`.

file

`deprecated/avm_global_analysis_ports.svh`

virtual

`no`

members

`static analysis_port #(T) s_analysis_ports[string]`

methods

`static function analysis_port #(T) get_analysis_port(string name)`

t1m_*_imp

Table 3-5 lists the t1m_*_imp deprecated in AVM 3.0

Table 3-5. Deprecated Implementations

Implementation	Interface
t1m_blocking_get_imp	t1m_blocking_get_if
t1m_blocking_get_peek_imp	t1m_blocking_get_peek_if
t1m_blocking_master_imp	t1m_blocking_master_if
t1m_blocking_peek_imp	t1m_blocking_peek_if
t1m_blocking_put_imp	t1m_blocking_put_if
t1m_blocking_slave_imp	t1m_blocking_slave_if
t1m_get_imp	t1m_get_if
t1m_get_peek_imp	t1m_get_peek_if
t1m_master_imp	t1m_master_if
t1m_nonblocking_get_imp	t1m_nonblocking_get_if
t1m_nonblocking_get_peek_imp	t1m_nonblocking_get_peek_if
t1m_nonblocking_master_imp	t1m_nonblocking_master_if
t1m_nonblocking_peek_imp	t1m_nonblocking_peek_if
t1m_nonblocking_put_imp	t1m_nonblocking_put_if
t1m_nonblocking_slave_imp	t1m_nonblocking_slave_if
t1m_peek_imp	t1m_peek_if
t1m_put_imp	t1m_put_if
t1m_slave_imp	t1m_slave_if
t1m_transport_imp	t1m_transport_if

Classes for Channels

The AVM supplies a FIFO channel and a variety of interfaces to access it. The interfaces have both blocking and nonblocking forms. Because SystemVerilog does not support multiple inheritance, the FIFO has a collection of “imps” implementations of abstract interfaces that are

used to access the FIFO. The FIFO is a named component and thus has a name and a location in the component hierarchy.

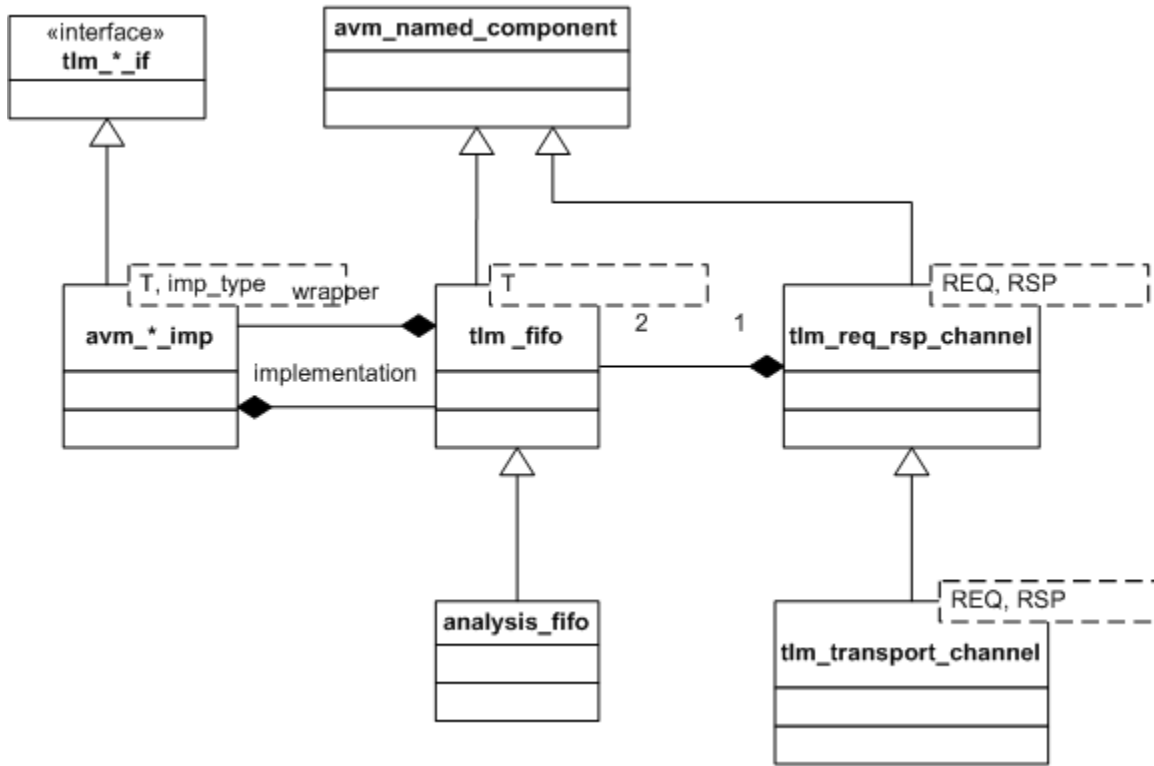


Figure 3-4. UML Diagram for Channels

analysis_fifo

```
 #(type T=int) extends tlm_fifo #(T)
```

An `analysis_fifo` is a `tlm_fifo` with an unbounded size and a `write()` interface. It can be used any place an `avm_subscriber` is used. Typical usage is as a buffer between an `analysis_port` in a monitor and an analysis component (i.e., a component derived from `avm_subscriber`).

file:

```
 tlm/tlm_fifos.svh
```

virtual

```
 no
```

parameters

```
 type T = int  
     Type of transactions to be stored in the FIFO.
```

members

```
 avm_analysis_imp #(T, analysis_fifo #(T)) analysis_export  
     analysis_export provides the write method to other components. Calling  
     ap.write(t) on a port bound to this export is the normal mechanism for writing to  
     an analysis FIFO.
```

methods

```
 function new(string name, avm_named_component parent=null)  
     This is the standard AVM 3.0 avm_named_component constructor. name is the local  
     name of this component. parent should be left unspecified when this component is  
     instantiated in statically elaborated constructs and must be specified when this  
     component is a child of another AVM component.
```

```
 function void write(input T t)  
     Transfers transaction t into the unbounded FIFO, which is guaranteed to succeed.
```

tlm_fifo

#(type T=int) extends avm_named_component
 tlm_fifo is a FIFO that implements all the unidirectional TLM interfaces.

file

tlm/tlm_fifos.svh

virtual

no

parameters

type **T** = int
Type of transactions to be stored in the FIFO.

members

```
typedef tlm_fifo #(T) this_type
```

```
avm_blocking_get_imp #(T, this_type) blocking_get_export  
avm_blocking_get_peek_imp #(T, this_type)
```

```
    blocking_get_peek_export
```

```
avm_blocking_peek_imp #(T, this_type) blocking_peek_export
```

```
avm_blocking_put_imp #(T, this_type) blocking_put_export
```

```
avm_get_imp #(T, this_type) get_export
```

```
avm_get_peek_imp #(T, this_type) get_peek_export
```

```
avm_nonblocking_get_imp #(T, this_type) nonblocking_get_export
```

```
avm_nonblocking_get_peek_imp #(T, this_type)
```

```
    nonblocking_get_peek_export
```

```
avm_nonblocking_peek_imp #(T, this_type) nonblocking_peek_export
```

```
avm_nonblocking_put_imp #(T, this_type) nonblocking_put_export
```

```
avm_peek_imp #(T, this_type) peek_export
```

```
avm_put_imp #(T, this_type) put_export
```

The implementations above export the relevant TLM interface. Every unidirectional TLM interface is implemented in `tlm_fifo` and exported using an appropriately named export.

```
avm_analysis_port #(T) put_ap
```

Analysis port to which the transaction is published whenever `put()` or `try_put()` succeeds.

```
analysis_port #(T) get_ap
```

Analysis port to which the transaction is published whenever `get()`, `try_get()`, `peek()`, or `try_peek()` succeeds.

```
local mailbox #(T) m
```

The internal mailbox used to implement the basic FIFO functionality.

```
local int m_size
```

`m_size` is the maximum size of the FIFO. A value of zero indicates no upper bound.

internal members

local int **m_pending_blocked_gets**

Used to calculate the result of `can_get()`. It should not be accessed by normal user code.

methods

function **new**(string name, avm_named_component parent=null,
int size=1)

name and parent are the normal AVM 3.0 constructor arguments. parent should be null if the t1m_fifo is going to be used in a statically elaborated construct. If it is defined within an avm_env, parent must be specified. size indicates the maximum size of the FIFO; a value of zero indicates no upper bound.

function bit **can_get**()

can_get() returns 1 if try_get() will be successful, and it returns 0 otherwise.

function bit **can_peek**()

can_peek() returns 1 if try_peek() will be successful, and it returns 0 otherwise.

function bit **can_put**()

can_put() returns 1 if try_put() will be successful, and it returns 0 otherwise.

function void **flush**()

flush() flushes the FIFO.

task **get**(output T t)

Does a blocking get and then publishes the gotten transaction using get_ap. Succeeds when there is something in the FIFO available to be gotten. get() is consuming. When it succeeds, t is no longer in the FIFO.

task **peek**(output T t)

Does a blocking peek() and then publishes the peeked transaction using get_ap. Succeeds when there is something in the FIFO available to be peeked. peek() is not consuming. When it succeeds, t is still in the FIFO.

task **put**(input T t)

Inserts transaction t to the internal mailbox and publishes the transaction to the put_ap when it is successful. Succeeds when there is room in the FIFO.

function int **size**()

This returns m_size.

function bit **try_get**(output T t)

Will get a transaction from the FIFO. If the FIFO contains a transaction, then it publishes the transaction across get_ap and returns 1. Otherwise, it returns 0. try_get() is consuming. When it succeeds, t is no longer in the FIFO.

function bit **try_peek**(output T t)

Will get a transaction from the FIFO if it contains a transaction, then it publishes the transaction across `get_ap` and returns 1. Otherwise, it returns 0. `peek()` is not consuming. When it succeeds, `t` is still in the FIFO.

```
function bit try_put(input T t)
```

Will put `t` into the FIFO if there is room, then publish the transaction across `put_ap` and return 1. Otherwise, it returns 0.

tlm_req_rsp_channel

```

    #(type REQ=int, type RSP=int)
    extends avm_named_component

```

tlm_req_rsp_channel contains a request FIFO of type REQ and a response FIFO of type RSP. These FIFOs can be of any size. This channel is particularly useful for dealing with pipelined protocols where the request and response are not tightly coupled.

file

```
tlm/tlm_req_rsp.svh
```

virtual

```
no
```

parameters

```

type REQ = int
    Type of transactions to be passed to/from the request FIFO.

type RSP = int
    Type of transactions to be passed to/from the response FIFO.

```

members

```

typedef tlm_req_rsp_channel #(REQ, RSP) this_type
protected tlm_fifo #(REQ) m_request_fifo
    The internal FIFO that stores the REQs.

protected tlm_fifo #(RSP) m_response_fifo
    The internal FIFO that stores the RSPs.

avm_blocking_put_export #(REQ) blocking_put_request_export
avm_nonblocking_put_export #(REQ)
    nonblocking_put_request_export
avm_put_export #(REQ) put_request_export
    The exports make the put, blocking put, and nonblocking put interfaces of the
    request FIFO externally visible. Through these interfaces, a master can put requests
    into the request FIFO.

avm_blocking_get_peek_export #(REQ)
    blocking_get_peek_request_export
avm_blocking_get_export #(REQ) blocking_get_request_export
avm_blocking_peek_export #(REQ) blocking_peek_request_export
avm_get_peek_export #(REQ) get_peek_request_export
avm_get_export #(REQ) get_request_export
avm_nonblocking_get_peek_export #(REQ)
    nonblocking_get_peek_request_export
avm_nonblocking_get_export #(REQ)
    nonblocking_get_request_export
avm_nonblocking_peek_export #(REQ)
    nonblocking_peek_request_export
avm_peek_export #(REQ) peek_request_export

```

These nine request get exports export the blocking, nonblocking, and combined get, peek, and get_peek interfaces of the request FIFO. These allow slaves to get or peek requests from the request FIFO.

```
avm_blocking_put_export #(RSP) blocking_put_response_export  
avm_nonblocking_put_export #(RSP)  
    nonblocking_put_response_export  
avm_put_export #(RSP) put_response_export
```

These three response put exports export the put, blocking put, and nonblocking put interfaces of the response FIFO. These allow a slave to put responses into the response FIFO.

```
avm_blocking_get_peek_export #(RSP)  
    blocking_get_peek_response_export  
avm_blocking_get_export #(RSP) blocking_get_response_export  
avm_blocking_peek_export #(RSP) blocking_peek_response_export  
avm_get_peek_export #(RSP) get_peek_response_export  
avm_get_export #(RSP) get_response_export  
avm_nonblocking_get_peek_export #(RSP)  
    nonblocking_get_peek_response_export  
avm_nonblocking_get_export #(RSP)  
    nonblocking_get_response_export  
avm_nonblocking_peek_export #(RSP)  
    nonblocking_peek_response_export  
avm_peek_export #(RSP) peek_response_export
```

These nine response get exports export the blocking, nonblocking, and combined get, peek and get_peek interfaces of the request FIFO. These allow masters to get or peek responses from the response FIFO.

```
avm_analysis_port #(RSP) response_ap  
    response_ap publishes an RSP whenever a put() or try_put() to the response  
    FIFO succeeds.
```

```
avm_analysis_port #(REQ) request_ap  
    Publishes a REQ whenever a put() or try_put() to the request FIFO succeeds.
```

```
avm_master_imp #(REQ, RSP, this_type,  
    tlm_fifo #(REQ), tlm_fifo #(RSP)) master_export  
    Exports a single interface that allows a master to put requests and get or peek  
    responses.
```

```
avm_slave_imp #(REQ, RSP, this_type  
    tlm_fifo #(REQ), tlm_fifo #(RSP)) slave_export  
    Exports a single interface that allows a slave to get or peek requests and put  
    responses.
```

```
avm_blocking_master_imp #(REQ, RSP, this_type,  
    tlm_fifo #(REQ), tlm_fifo #(RSP)) blocking_master_export  
    Exports a single blocking interface that allows a master to put requests and get or  
    peek responses.
```

```
avm_blocking_slave_imp #(REQ, RSP, this_type,  
    tlm_fifo #(REQ), tlm_fifo #(RSP)) blocking_slave_export
```

Exports a single blocking interface that allows a slave to get or peek requests and put responses.

```
avm_nonblocking_master_imp #(REQ, RSP, this_type,  
  tlm_fifo #(REQ), tlm_fifo #(RSP)) nonblocking_master_export
```

Exports a single nonblocking interface that allows a master to put requests and get or peek responses.

```
avm_nonblocking_slave_imp #(REQ, RSP, this_type,  
  tlm_fifo #(REQ), tlm_fifo #(RSP)) nonblocking_slave_export
```

Exports a single nonblocking interface that allows a slave to get or peek requests and put responses.

methods

```
function new(string name, avm_named_component parent=null,  
  int request_fifo_size=1,  
  int response_fifo_size = 1)
```

`name` and `parent` are the standard AVM 3.0 constructor arguments. `parent` must be null if this component is defined within a static component such as a module, program block, or interface, and it must take a non value if it is defined inside an `avm_env`. The last two arguments specify the request and response FIFO sizes, which have default values of one.

internal methods:

```
function void create_master_slave_exports()  
  Creates the bidirectional exports for both master and slave.
```

```
function void create_response_exports()  
  Creates the unidirectional request exports for both master and slave.
```

```
function void export_response_connections()  
  Connects the response FIFO to the appropriate exports.
```

```
function void export_request_connections()  
  Connects the request FIFO to the appropriate exports.
```

tlm_transport_channel

```
 #(type REQ=int, type RSP=int)  
 extends tlm_req_rsp_channel #(REQ, RSP)
```

A `tlm_transport_channel` is a `tlm_req_rsp_channel` that implements the transport interface. It is useful when modeling a nonpipelined bus at the transaction level. Because the requests and responses have a tightly coupled one-to-one relationship, the request and response FIFO sizes must be one.

file

```
 tlm/tlm_req_rsp.svh
```

virtual

```
 no
```

parameters

```
 type REQ = int  
   Type of transactions to be passed to/from the request FIFO.
```

```
 type RSP = int  
   Type of transactions to be passed to/from the response FIFO.
```

members

```
 typedef tlm_transport_channel #(REQ, RSP) this_type  
 avm_transport_imp #(REQ, RSP, this_type) transport_export  
   The mechanism by which external components gain access to the transport() task.
```

methods

```
 function new(string name, avm_named_component parent=null)  
   name and parent are the standard AVM 3.0 constructor arguments. parent must be  
   null if this component is defined within a statically elaborated construct such as a  
   module, program block, or interface, and it must take a non-null value if it is defined  
   inside an avm_env.
```

```
 task transport(input REQ request, output RSP response)  
   Calls put(request) followed by get(response).
```

TLM Interfaces

The TLM interfaces are a collection of pure virtual classes that define the way transaction objects move between components. Each interface class supplies a set of one or more tasks and function prototypes. Interface implementations (imps), ports, exports, and channels use the TLM interfaces to define the set of functions and tasks that each needs to implement.

analysis_if

 #(type T=int)

file

tlm/tlm_ifs.svh

The analysis interface is a nonblocking, non-negotiable, unidirectional interface. It is typically used to transfer a transaction from a monitor, which cannot block, to a scoreboard or coverage object.

virtual

yes

parameters

type **T** = int
Type of transactions to be handled by this interface.

members

<none>

methods

Pure virtual methods must have an implementation specified in a subclass.

pure virtual function void **write**(input **T** **t**)
Takes transaction **t**, operates on it (e.g., copies it, records values for functional coverage, etc.) in some nonblocking way and returns immediately.

tlm_blocking_get_if

`#(type T=int)`
The blocking get interface

file

`tlm/tlm_ifs.svh`

virtual

`yes`

parameters

`type T = int`
Type of transactions to be handled by this interface.

members

`<none>`

methods

Pure virtual methods must have an implementation specified in a subclass.

`pure virtual task get(output T t)`
Blocks until the callee is able to supply a transaction `t`. This is a consuming method, so subsequent calls to `get()` return a different transaction (or a new copy of the same transaction).

tlm_blocking_get_peek_if

```
 #(type T=int)  
 The blocking get interface
```

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int  
     Type of transactions to be handled by this interface.
```

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual task get( output T t )  
     Blocks until the callee is able to supply a transaction t. This is a consuming method,  
     so subsequent calls to get() return a different transaction (or a new copy of the  
     same transaction).
```

```
 pure virtual task peek( output T t )  
     Blocks until the callee is able to supply a transaction. This is a nonconsuming  
     method, so subsequent calls to peek() or the next call to get() return the same  
     transaction.
```

tlm_blocking_peek_if

`#(type T=int)`
The blocking peek interface

file

`tlm/tlm_ifs.svh`

virtual

`yes`

parameters

`type T = int`
Type of transactions to be handled by this interface.

members

`<none>`

methods

Pure virtual methods must have an implementation specified in a subclass.

`pure virtual task peek(output T t)`
Blocks until the callee is able to supply a transaction. This is a nonconsuming method, so subsequent calls to `peek()` or the next call to `get()` return the same transaction.

tlm_blocking_put_if

```
 #(type T=int)  
 The blocking put interface
```

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int  
   Type of transactions to be handled by this interface.
```

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual task put( input T t );  
   Blocks until the callee is able to accept a transaction.
```

tlm_blocking_master_if

```
 #(type REQ = int, type RSP = int)  
 The blocking master interface
```

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int  
   Type of transactions to be handled by this interface.
```

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual task put( input REQ req )  
   Blocks until the callee is able to accept a request transaction.
```

```
 pure virtual task get( output RSP rsp )  
   Blocks until the callee is able to supply a response transaction t. This is a consuming  
   method, so subsequent calls to get() return a different transaction (or a new copy of  
   the same transaction).
```

```
 pure virtual task peek( output RSP rsp )  
   Blocks until the callee is able to supply a response transaction. This is a  
   nonconsuming method, so subsequent calls to peek() or the next call to get()  
   return the same transaction.
```

tlm_blocking_slave_if

```
 #(type T=int)  
 The blocking slave interface
```

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int  
   Type of transactions to be handled by this interface.
```

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual task put( RSP rsp )  
   Blocks until the callee is able to accept a response transaction.
```

```
 pure virtual task get( output REQ req )  
   Blocks until the callee is able to supply a request transaction  $t$ . This is a consuming  
   method, so subsequent calls to get() return a different transaction (or a new copy of  
   the same transaction).
```

```
 pure virtual task peek( output REQ req );  
   Blocks until the callee is able to supply a request transaction. This is a  
   nonconsuming method, so subsequent calls to peek() or the next call to get()  
   return the same transaction.
```

tlm_get_if

```
 #(type T=int)
```

The get interface is a unidirectional consuming interface. It has both functions, which cannot block, and tasks, which may block.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int
```

Type of transactions to be handled by this interface.

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_get()
```

Returns 1 if a callee can supply a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `get()` or `try_get()` is guaranteed to succeed.

```
 pure virtual task get(output T t)
```

Blocks until the callee is able to supply a transaction `t`. This is a consuming method, so subsequent calls to `get()` return a different transaction (or a new copy of the same transaction).

```
 pure virtual function bit try_get(output T t)
```

Returns immediately and supplies a transaction `t`, if one is available. If successful, will return 1, otherwise will return 0 (and `t` will be undefined).

tlm_get_peek_if

```
 #(type T=int)
```

The get peek interface is a unidirectional interface. It has both blocking tasks and nonblocking functions, and it has both consuming and nonconsuming methods.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int
```

Type of transactions to be handled by this interface.

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_get()
```

Returns 1 if a callee can supply a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `get()`, `try_get()`, `peek()`, or `try_peek()` is guaranteed to succeed.

```
 pure virtual function bit can_peek()
```

Returns 1 if a callee can supply a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `get()`, `try_get()`, `peek()`, or `try_peek()` is guaranteed to succeed.

```
 pure virtual task get(output T t)
```

Blocks until the callee is able to supply a transaction `t`. This is a consuming method, so subsequent calls to `get()` return a different transaction (or a new copy of the same transaction).

```
 pure virtual task peek(output T t)
```

Blocks until the callee is able to supply a transaction. This is a nonconsuming method, so subsequent calls to `peek()` or the next call to `get()` return the same transaction.

```
 pure virtual function bit try_get(output T t)
```

Returns immediately and supplies a transaction τ , if one is available. If successful, it returns 1 (and τ will no longer be available). Otherwise, it returns 0 (and τ will be undefined).

pure virtual function bit **try_peek(output T τ)**

Returns immediately and supplies a transaction τ , if one is available. If successful, returns 1 (and τ will still be available). Otherwise, it returns 0 (and τ is undefined).

tlm_master_if

```
 #(type REQ=int, type RSP=int)
```

The master interface is a bidirectional interface. It enables a master to put requests and get or peek responses. It contains both blocking and nonblocking methods.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type REQ = int  
     Type of transactions to be handled on the put side.
```

```
 type RSP = int  
     Type of transactions to be handled on the get/peek side.
```

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_get()  
     Returns 1 if a callee can supply a response transaction and 0 otherwise. If no time  
     elapses and nothing else happens to modify the state of the underlying component,  
     then a subsequent call to get(), try_get(), peek(), or try_peek() is guaranteed  
     to succeed.
```

```
 pure virtual function bit can_peek()  
     Returns 1 if a callee can supply a response transaction and 0 otherwise. If no time  
     elapses and nothing else happens to modify the state of the underlying component,  
     then a subsequent call to get(), try_get(), peek(), or try_peek() is guaranteed  
     to succeed.
```

```
 pure virtual function bit can_put()  
     Returns 1 if the callee can accept a request and 0 otherwise. If no time elapses and  
     nothing else happens to modify the state of the underlying component, then a  
     subsequent call to put() is guaranteed to succeed.
```

```
 pure virtual task get(output RSP rsp)  
     Blocks until the callee is able to supply a response transaction, rsp. This is a  
     consuming method, so subsequent calls to get() return a different transaction (or a  
     new copy of the same transaction).
```

pure virtual task **peek(output RSP rsp)**

Blocks until the callee is able to supply a response transaction. This is a nonconsuming method, so subsequent calls to `peek()` or the next call to `get()` returns the same transaction.

pure virtual task **put(REQ req)**

Blocks until the callee is able to accept a request transaction, `req`.

pure virtual function bit **try_get(output RSP rsp)**

Returns immediately and supplies a response transaction `rsp`, if one is available. If successful, it returns 1 (and `rsp` will no longer be available). Otherwise, it returns 0 (and `rsp` will be undefined).

pure virtual function bit **try_peek(output RSP rsp)**

Returns immediately and supplies a response transaction `rsp`, if one is available. If successful, it returns 1 (and `rsp` will still be available). Otherwise, it returns 0 (and `rsp` will be undefined).

pure virtual function bit **try_put(REQ req)**

Returns immediately. If the callee can accept a request, it returns 1, otherwise, it returns 0.

tlm_nonblocking_get_if

```
 #(type T=int)
```

The nonblocking get interface is a nonblocking, unidirectional, consuming interface.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int
```

Type of transactions to be handled by this interface.

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_get()
```

Returns 1 if a callee can supply a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `get()` or `try_get()` is guaranteed to succeed.

```
 pure virtual function bit try_get(output T t)
```

Returns immediately and supplies a transaction `t`, if one is available. If successful, it returns 1. Otherwise, it returns 0 (and `t` will be undefined).

tlm_nonblocking_get_peek_if

`#(type T=int)`

The nonblocking get peek interface is a nonblocking, unidirectional, interface.

file

`tlm/tlm_ifs.svh`

virtual

`yes`

parameters

`type T = int`

Type of transactions to be handled by this interface.

members

`<none>`

methods

Pure virtual methods must have an implementation specified in a subclass.

`pure virtual function bit can_get()`

Returns 1 if a callee can supply a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `get()` or `try_get()` is guaranteed to succeed.

`pure virtual function bit can_peek()`

Returns 1 if a callee can supply a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `try_get()` or `try_peek()` is guaranteed to succeed.

`pure virtual function bit try_get(output T t)`

Returns immediately and supplies a transaction `t`, if one is available. If successful, it returns 1. Otherwise, it returns 0 (and `t` will be undefined).

`pure virtual function bit try_peek(output T t)`

Returns immediately and supplies a transaction `t`, if one is available. If successful, it returns 1 (and `t` will still be available). Otherwise, it returns 0 (and `t` will be undefined).

tlm_nonblocking_master_if

```
 #(type REQ=int, type RSP=int)
```

The nonblocking master interface is a bidirectional interface. It enables a master to put requests and get or peek responses.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type REQ = int  
     Type of transactions to be handled on the put side.
```

```
 type RSP = int  
     Type of transactions to be handled on the get/peek side.
```

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_get()  
     Returns 1 if a callee can supply a response transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to get(), try_get(), peek(), or try_peek() is guaranteed to succeed.
```

```
 pure virtual function bit can_peek()  
     Returns 1 if a callee can supply a response transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to get(), try_get(), peek(), or try_peek() is guaranteed to succeed.
```

```
 pure virtual function bit can_put()  
     Returns 1 if the callee can accept a request and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to put() is guaranteed to succeed.
```

```
 pure virtual function bit try_get(output RSP rsp)  
     Returns immediately and supplies a response transaction rsp, if one is available. If successful, it returns 1 (and rsp will no longer be available). Otherwise, it returns 0 (and rsp will be undefined).
```

pure virtual function bit **try_peek(output RSP rsp)**

Returns immediately and supplies a response transaction `rsp`, if one is available. If successful, it returns 1 (and `rsp` will still be available). Otherwise, it returns 0 (and `rsp` will be undefined).

pure virtual function bit **try_put(REQ req)**

Returns immediately. If the callee can accept a request, it returns 1. Otherwise, it returns 0.

tlm_nonblocking_peek_if

```
 #(type T=int)
```

The nonblocking peek interface is a unidirectional, nonblocking, nonconsuming interface.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int
```

Type of transactions to be handled by this interface.

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_peek()
```

Returns 1 if a callee can supply a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `try_get()` or `try_peek()` is guaranteed to succeed.

```
 pure virtual function bit try_peek(output T t)
```

Returns immediately and supplies a transaction `t`, if one is available. If successful, it returns 1 (and `t` will still be available). Otherwise, it returns 0 (and `t` will be undefined).

tlm_nonblocking_put_if

```
 #(type T=int)
```

The nonblocking put interface is a unidirectional, nonblocking interface.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int
```

Type of transactions to be handled by this interface.

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_put()
```

Returns 1 if the callee can accept a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `try_put()` is guaranteed to succeed.

```
 pure virtual function bit try_put(T t)
```

Returns immediately. If the callee can accept a transaction, it returns 1. Otherwise, it returns 0.

tlm_nonblocking_slave_if

```
 #(type REQ=int, type RSP=int)
```

The nonblocking slave interface is a bidirectional nonblocking interface. It allows a slave to get or peek requests and put responses.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type REQ = int  
     Type of transactions to be handled on the get/peek side.  
  
 type RSP = int  
     Type of transactions to be handled on the put side.
```

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_get()  
     Returns 1 if a callee can supply a request transaction and 0 otherwise. If no time  
     elapses and nothing else happens to modify the state of the underlying component,  
     then a subsequent call to get(), try_get(), peek(), or try_peek() is guaranteed  
     to succeed.  
  
 pure virtual function bit can_peek()  
     Returns 1 if a callee can supply a request transaction and 0 otherwise. If no time  
     elapses and nothing else happens to modify the state of the underlying component,  
     then a subsequent call to get(), try_get(), peek(), or try_peek() is guaranteed  
     to succeed.  
  
 pure virtual function bit can_put()  
     Returns 1 if the callee can accept a response and 0 otherwise. If no time elapses and  
     nothing else happens to modify the state of the underlying component, then a  
     subsequent call to put() is guaranteed to succeed.  
  
 pure virtual function bit try_get(output REQ req)  
     Returns immediately and supplies a request transaction req, if one is available. If  
     successful, it returns 1 (and req will no longer be available). Otherwise, it returns 0  
     (and req is undefined).
```

pure virtual function bit **try_peek(output REQ req)**

Returns immediately and supplies a request transaction `req`, if one is available. If successful, it returns 1 (and `req` will still be available). Otherwise, it returns 0 (and `req` is undefined).

pure virtual function bit **try_put(RSP rsp)**

Returns immediately. If the callee can accept a response, it returns 1. Otherwise, it returns 0.

tlm_peek_if

```
 #(type T=int)
```

The peek interface is a unidirectional, nonconsuming interface.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int
```

Type of transactions to be handled by this interface.

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_peek()
```

Returns 1 if a callee can supply a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `get()`, `try_get()`, `peek()`, or `try_peek()` is guaranteed to succeed.

```
 pure virtual task peek(output T t)
```

Blocks until the callee is able to supply a transaction. This is a nonconsuming method, so subsequent calls to `peek()` or the next call to `get()` return the same transaction.

```
 pure virtual function bit try_peek(output T t)
```

Returns immediately and supplies a transaction `t`, if one is available. If successful, it returns 1 (and `t` will still be available). Otherwise, it returns 0 (and `t` is undefined).

tlm_put_if

```
 #(type T=int)
```

The put interface is a unidirectional interface. It contains both blocking tasks and nonblocking functions.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type T = int
```

Type of transactions to be handled by this interface.

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual task put(input T t)
```

The `put()` task blocks until the callee is able to accept a T.

```
 pure virtual function bit can_put()
```

Returns 1 if the callee can accept a transaction and 0 otherwise. If no time elapses and nothing else happens to modify the state of the underlying component, then a subsequent call to `try_put()` is guaranteed to succeed.

```
 pure virtual function bit try_put(T t)
```

Returns immediately. If the callee can accept a transaction, it returns 1, otherwise, it returns 0.

```
 pure virtual task put(T t)
```

Blocks until the callee is able to accept a transaction.

tlm_slave_if

```
 #(type REQ=int, type RSP=int)
```

The slave interface is a bidirectional interface. It allows a slave to get or peek requests and put responses in either blocking or nonblocking forms.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type REQ = int  
     Type of transactions to be handled on the get/peek side.  
  
 type RSP = int  
     Type of transactions to be handled on the put side.
```

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual function bit can_get()  
     Returns 1 if a callee can supply a request transaction and 0 otherwise. If no time  
     elapses and nothing else happens to modify the state of the underlying component,  
     then a subsequent call to get(), try_get(), peek(), or try_peek() is guaranteed  
     to succeed.  
  
 pure virtual function bit can_peek()  
     Returns 1 if a callee can supply a request transaction and 0 otherwise. If no time  
     elapses and nothing else happens to modify the state of the underlying component,  
     then a subsequent call to get(), try_get(), peek(), or try_peek() is guaranteed  
     to succeed.  
  
 pure virtual function bit can_put()  
     Returns 1 if the callee can accept a response and 0 otherwise. If no time elapses and  
     nothing else happens to modify the state of the underlying component, then a  
     subsequent call to put() is guaranteed to succeed.  
  
 pure virtual task get(output REQ req)  
     Blocks until the callee is able to supply a request transaction, req. This is a  
     consuming method, so subsequent calls to get() return a different transaction (or a  
     new copy of the same transaction).
```

- pure virtual task **peek(output REQ req)**
Blocks until the callee is able to supply a request transaction. This is a nonconsuming method, so subsequent calls to `peek()` or the next call to `get()` return the same transaction.
- pure virtual task **put(RSP rsp)**
Blocks until the callee is able to accept a response transaction, `rsp`.
- pure virtual function bit **try_get(output REQ req)**
Returns immediately and supplies a request transaction `req`, if one is available. If successful, it returns 1 (and `req` will no longer be available). Otherwise, it returns 0 (and `req` will be undefined).
- pure virtual function bit **try_peek(output REQ req)**
Returns immediately and supplies a request transaction `req`, if one is available. If successful, it returns 1 (and `req` will still be available). Otherwise, it returns 0 (and `req` will be undefined).
- pure virtual function bit **try_put(RSP rsp)**
Returns immediately. If the callee can accept a response, it returns 1. Otherwise, it returns 0.

tlm_transport_if

```
 #(type REQ=int, type RSP=int)
```

The transport interface is a bidirectional blocking interface. It is used when there is a tight one-to-one coupling between request and response, typically in the context of nonpipelined buses.

file

```
 tlm/tlm_ifs.svh
```

virtual

```
 yes
```

parameters

```
 type REQ = int  
     Type of transactions to be sent.
```

```
 type RSP = int  
     Type of transactions to be received.
```

members

```
 <none>
```

methods

Pure virtual methods must have an implementation specified in a subclass.

```
 pure virtual task transport(input REQ request,  
                             output RSP response)  
     Sends a request transaction to the callee and blocks until it obtains a response  
     transaction back from the callee.
```

Transactions

avm_built_in_clone

`#(type T=int)`

This policy class is used to clone built-in types. It is used to build generic components that will work with either classes or built-in types.

file

`vbase/avm_policies.svh`

virtual

`no`

parameters

`type T = int`

The return type of the `clone()` method.

members

`<none>`

methods

`static function T clone(input T from)`

Returns the value of `from`.

avm_built_in_comp

```
 #(type T=int)
```

This policy class is used to compare built-in types. It is used to build generic components that work with either classes or built-in types.

file

```
vbase/avm_policies.svh.
```

virtual

```
no
```

parameters

```
type T = int
```

The type of the items to be compared.

members

```
<none>
```

methods

```
static function bit comp(input T a, input T b)
```

Returns the value of a==b.

avm_built_in_converter

```
 #(type T=int)
```

This policy class is used to convert built-in types to strings. It is used to build generic components that will work with either classes or built-in types.

file

vbase/avm_policies.svh

virtual

no

parameters

```
type T = int  
    The type of the item to be converted.
```

members

<none>

methods

```
static function string convert2string(input T t);  
    Returns the value of t as a string.
```

avm_built_in_pair

```
 #(type T1=int, type T2=T1)
 extends avm_transaction
```

This class represents a pair of built in types.

file

```
utils/avm_pair.svh
```

virtual

```
no
```

parameters

```
type T1 = int
```

The type of the first element of the pair.

```
type T2 = T1
```

The type of the second element of the pair. By default, the two types are the same.

members

```
typedef avm_built_in_pair #(T1, T2) this_type
T1 first
```

The first element of the pair.

```
T2 second
```

The second element of the pair.

methods

```
virtual function string convert2string()
function bit comp(this_type t)
function void copy(input this_type t)
function avm_transaction clone()
```

Since `avm_built_in_pair` is a transaction class, it provides the four compulsory methods as defined by AVM 3.0.

avm_class_clone

`#(type T=int)`

This policy class is used to clone classes. It is used to build generic components that work with either classes or built-in types.

file

`vbase/avm_policies.svh`

virtual

`no`

members

`<none>`

methods

`static function avm_transaction clone(input T from)`

This method returns `from.clone()`.

avm_class_comp

#(type T=int)

This policy class is used to compare classes. It is used to build generic components that work with either built-in types or classes.

file

vbase/avm_policies.svh

virtual

no

members

<none>

methods

static function bit comp(input T a, input T b)

This method returns a.comp(b).

avm_class_converter

`#(type T=int)`

This policy class is used to convert classes to strings. It is used to build generic components that work with either built-in types or classes.

file

`vbase/avm_policies.svh`

virtual

`no`

members

`<none>`

methods

`static function string convert2string(input T t)`

This method returns `t.convert2string()`.

avm_class_pair

```
 #(type T1=int, type T2=T1)
 extends avm_transaction
```

This class represents a pair of classes.

file

```
utils/avm_pairs.svh
```

virtual

```
no
```

members

```
typedef avm_class_pair #(T1, T2) this_type
T1 first
```

This is the first element in the pair.

```
T2 second
```

This is the second element in the pair.

methods

```
function new(input T1 f=null, input T2 s=null)
```

A constructor, with optional arguments for first and second. No cloning is performed for nondefault values.

```
function string convert2string
function bit comp(this_type t)
function void copy(input this_type t)
function avm_transaction clone
```

Since `avm_built_in_pair` is a transaction class, it provides the four compulsory methods as defined by AVM 3.0.

avm_transaction

This is the base class for all AVM transactions.

file

vbase/avm_transaction.svh

virtual

yes

members

<none>

methods

pure virtual function avm_transaction clone

This virtual method returns a handle to a clone of this transaction. Since it is virtual, the clone is deep in relation to the inheritance hierarchy, although it may be shallow or deep in relation to members of subclasses that are themselves handles.

pure virtual function string convert2string

This method converts the transaction into a string. Since it is virtual, it is also deep, in relation to the inheritance hierarchy.

In addition to the two methods described above, any transaction T that is a subtype of `avm_transaction` must also define the following two methods.

function bit comp(input T t);

This function compares this transaction with t . It returns 1 if it is the same and 0 if they are different.

function void copy(input T t);

This function copies the contents of t into this transaction. It may be shallow or deep in relation to handles. It usually calls `super.copy(t)` if T is not a direct base class of `avm_transaction`.

Reporting

The reporting classes provide a facility for issuing reports with different severities and ids, and to different files. The primary interface to the reporting facility is `avm_report_client`.

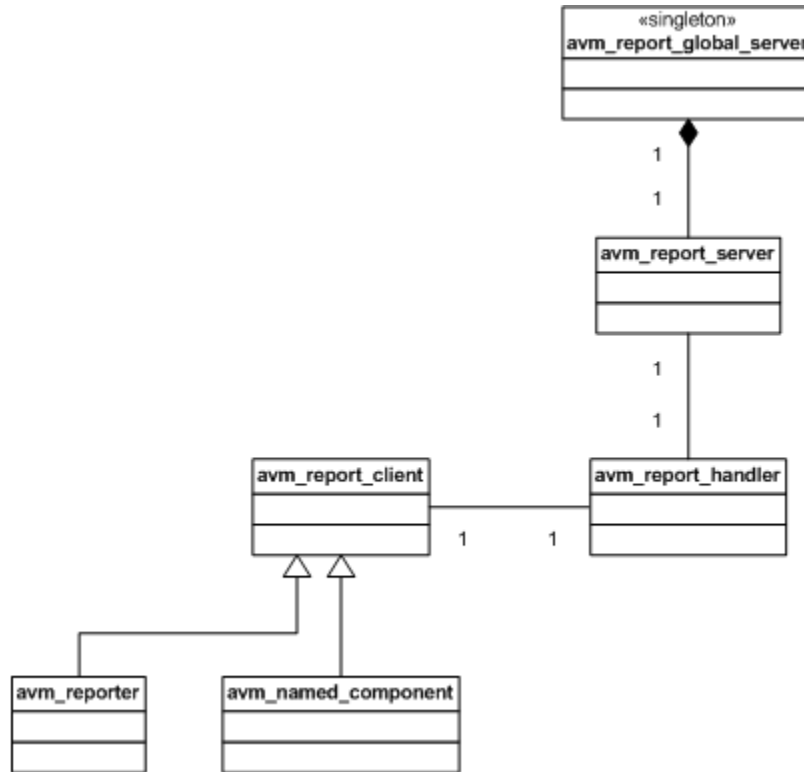


Figure 3-5. UML Diagram for Reporting Classes

avm_report_client

`avm_report_client` is a base class from which all components that want to use the AVM reporting facility must inherit. It provides methods to issue messages, change the action associated with these messages, associate files with messages, and execute hook methods as a result of these messages.

All of the state information relating to actions and files associated with different types of messages is held in an `avm_report_handler`. Most of the methods in this class are delegated to a report handler, which in turn delegates the actual formatting and production of messages to a central `avm_report_server`.

file

reporting/avm_report_client.svh

virtual

yes

members

protected `avm_report_handler m_rh`
Handle to a report handler, which stores all the state information about actions and files. It may be unique to this report client or shared with other clients.

local `string m_report_name`
The name of the report handler. This name is printed out at the beginning of each message.

methods

function `new(string name="")`
The constructor requires a name, and creates a new report handler that is unique to this client.

function void `avm_report_error(string id, string message, int verbosity_level=100, string filename="", int line=0)`
One of the four core reporting methods, it issues a report of severity ERROR. If the verbosity level of this report is higher than the maximum verbosity level of the report handler, this report is simply ignored. By default, a warning is displayed on the command line, logged in a file if one has been set, and counted. If the error count in any report handler exceeds its maximum quit count, then the `die()` method is called. The default verbosity level for an error is 100.

function void `avm_report_fatal(string id, string message, int verbosity_level=0, string filename="", int line=0)`
One of the four core reporting methods, it issues a report of severity FATAL. If the verbosity level of this report is higher than the maximum verbosity level of the

report handler, this report is simply ignored. By default, a fatal error is displayed on the command line, and then it calls the `die()` method. The default verbosity level for a fatal report is 0.

```
function void avm_report_message(string id, string message,
                                int verbosity_level=300,
                                string filename="", int line=0)
```

One of the four core reporting methods, it issues a report of severity MESSAGE. If the verbosity level of this report is higher than the maximum verbosity level of the report handler, this report is simply ignored. By default, a message is displayed on the command line and logged in a file, if one has been set. The default verbosity level for a message is 300.

```
function void avm_report_warning(string id, string message,
                                  int verbosity_level=200,
                                  string filename="", int line=0)
```

One of the four core reporting methods, it issues a report of severity WARNING. If the verbosity level of this report is higher than the maximum verbosity level of the report handler, this report is simply ignored. By default a warning is displayed on the command line and logged in a file, if one has been set. The default verbosity level for a warning is 200.

```
function avm_report_handler get_report_handler()
```

Provides public access to the report handler, which stores all the state information.

```
function string get_report_name()
```

Provides public access to the report name.

```
virtual function void report_header(FILE f=0)
```

Prints version and copyright information. This information will be sent to the command line if `f` is 0, or to the file descriptor `f` if it is not 0. This method is called by `avm_env` immediately after the construction phase and before the connect phase.

```
virtual function bit report_hook(string id, string message,
                                   int verbosity,
                                   string filename, int line)
```

Called only if the `CALL_HOOK` bit is specified in the action associated with the report. By default, it does nothing other than return 1, but it can be overloaded in a subclass. If this method returns 0, the report will not be processed by the report server.

```
virtual function bit report_error_hook(string id,
                                         string message,
                                         int verbosity,
                                         string filename, int line)
```

Called only if the `CALL_HOOK` bit is specified in the action associated with an error report. By default, it does nothing other than return 1, but it can be overloaded in a subclass. If this method returns 0, the error will not be processed by the report server.

```
virtual function bit report_fatal_hook(string id,
                                         string message,
                                         int verbosity,
```

```
string filename, int line)
```

Called only if the `CALL_HOOK` bit is specified in the action associated with a fatal report. By default, it does nothing other than return 1, but it can be overloaded in a subclass. If this method returns 0, the fatal will not be processed by the report server.

```
virtual function bit report_message_hook(string id,  
string message,  
int verbosity,  
string filename, int line)
```

Called only if the `CALL_HOOK` bit is specified in the action associated with a message. By default, it does nothing other than return 1, but can be overloaded in a subclass. If this method returns 0, the message will not be processed by the report server.

```
function void report_summarize(FILE f=0)
```

Produces statistical information on the reports issued by the central report server. This information will be sent to the command line if `f` is 0, or to the file descriptor `f` if it is not 0.

```
virtual function bit report_warning_hook(string id,  
string message,  
int verbosity,  
string filename, int line)
```

Called only if the `CALL_HOOK` bit is specified in the action associated with a warning. By default, it does nothing other than return 1, but can be overloaded in a subclass. If this method returns 0, the warning will not be processed by the report server.

```
function void reset_report_handler()
```

Reinitializes the client's report handler to the default settings.

```
function void set_report_handler(avm_report_handler hndlr)
```

Sets the report handler, thus allowing more than one client to share the same report handler.

```
function void set_report_max_quit_count(int m)
```

Sets the value of the `max_quit_count` in the report handler to `m`. When the number of `COUNT` actions reaches `m`, the `die()` method is called. The default value of 0 indicates that there is no upper limit to the number of `COUNT`ed reports.

```
function void set_report_name(string s)
```

Sets the report name.

```
function void set_report_severity_action (severity s,  
action a)
```

Sets the action associated with a severity. An action can take the value `NO_ACTION` (`5'b00000`) or can be composed of the bitwise OR of any combination of `DISPLAY`, `LOG`, `COUNT`, `EXIT`, or `CALL_HOOK`.

```
function void set_report_verbosity_level(int verbosity_level)
```

Sets the maximum verbosity level for the client's report handler. If the verbosity of any report exceeds this maximum value, then the report is ignored.

```
function void set_report_id_action (string id, action a)
```

This method sets the action associated with an id. An action associated with an id takes priority over an action associated with a severity. An action can take the value `NO_ACTION` (`5'b00000`) or can be composed of the bitwise OR of any combination of `DISPLAY`, `LOG`, `COUNT`, `EXIT`, or `CALL_HOOK`.

```
function void set_report_severity_id_action (severity s,
                                             string id,
                                             action a)
```

This method sets the action associated with a (severity,id) pair. An action associated with a (severity,id) pair takes priority over an action associated with either the severity or the id alone. An action can take the value `NO_ACTION` (`5'b00000`) or can be composed of the bitwise OR of any combination of `DISPLAY`, `LOG`, `COUNT`, `EXIT`, or `CALL_HOOK`.

```
function void set_report_default_file (input FILE f)
```

This method sets the file descriptor associated by default with any report issued by this client's report handler. The default value is 0, which means that even if the action includes a `LOG` attribute, the report is not sent to a file.

```
function void set_report_severity_file (severity s,
                                         FILE f)
```

This method sets the file descriptor associated with a severity. A file descriptor associated with a severity takes priority over the default file descriptor.

```
function void set_report_id_file (input string id, input FILE f)
```

This method sets the file descriptor associated with an id. A file descriptor associated with an id takes priority over the default file descriptor and a file descriptor associated with a severity.

```
function void set_report_severity_id_file (severity s,
                                             string id,
                                             FILE f)
```

This method sets the file descriptor associated with a (severity,id) pair. A file descriptor associated with a (severity,id) pair takes priority over the default file descriptor, a file descriptor associated with a severity, or a file descriptor associated with an id.

```
function void dump_report_state()
```

This method dumps the internal state of the report handler. This includes information about the maximum quit count, the maximum verbosity, and the action and files associated with severities, ids, and (severity,id) pairs.

```
virtual function void die()
```

This method is called by the report server if a report reaches the maximum quit count or has an `EXIT` action associated with it (this is part of the default action for a fatal error).

If this method is called in a client that is actually a named component defined in an `avm_env`, then all the `avm_env`'s `run()` tasks are killed and the `avm_env` goes through the report phase, which by default, calls `report_summarize()`. In this case, any other `avm_env`'s in the simulation will not be affected.

If `die()` is called in a report client that is not an `avm_named_component`, or in an `avm_named_component` defined outside of an `avm_env`, then `report_summarize()` is called and the simulation terminates with `$finish`.

avm_report_handler

`avm_report_handler` is the class to which many of the methods in `avm_report_client` are delegated. None of its methods are intended to be called directly from normal test bench code.

It stores the maximum verbosity, actions, and files that affect the way reports are handled. The relationship between report clients and report handlers is usually one to one, but it can, in theory, be many to one. If a report needs processing, it passes it on to the central report server. The relationship between report handlers and report servers is many to one.

file

`reporting/avm_report_handler.svh`

virtual

no

members

`avm_report_server m_srvr`

This is the central report server that actually processes the reports.

`int m_max_verbosity_level`

This is the maximum verbosity of reports that this report handler forwards to the report server. The default value is 10000.

`action severity_actions[severity]`

This is the array that contains the actions associated with each severity. The default values are given by the table below.

Severity	Actions
MESSAGE	DISPLAY
WARNING	DISPLAY
ERROR	DISPLAY COUNT
FATAL	DISPLAY EXIT

`id_actions_array id_actions`

This is the array of actions associated with each string id. By default, there are no entries in this array.

`id_actions_array severity_id_actions[severity]`

This is an associative array of associative arrays. If it exists, then `severity_id_actions[s][i]` contains the actions associated with the (severity,id) pair (s,i). By default, there are no entries in this array.

FILE `default_file_handle`
This is the default file handle for this report handler. By default, it is set to 0, which means that reports are not sent to a file even if a LOG attribute is set in the action associated with the report.

FILE `severity_file_handles[severity]`
This array contains the file handle associated with each severity.

id_file_array `id_file_handles`
This array contains the file handle associated with each string id.

id_file_array `severity_id_file_handles[severity]`
This associative array of associative arrays contains the file descriptor associated with each (severity,id) pair, if there are any.

methods

function `new()`
The constructor.

function void `set_max_quit_count(int m)`
See `avm_report_client::set_report_max_quit_count` (see page 876).

function void `summarize(FILE f=0)`
See `avm_report_client::report_summarize` (see page 876).

function void `report_header(FILE f=0)`
See `avm_report_client::report_header` (see page 875).

function void `initialize()`
This method is called by the constructor to initialize the arrays and other variables described above to their default values.

virtual function bit `run_hooks(avm_report_client client, severity s, string id, string message, int verbosity, string filename, int line)`
`run_hooks` is called if the CALL_HOOK attribute is set for this report. It calls the client's `report_hook` and severity specific hook method. If either returns 0, then the report is not processed.

local function FILE `get_severity_id_file(severity s, string id)`
This method looks up the file descriptor associated with reports with this severity and id.

function void `set_verbosity_level(int verbosity_level)`
See `avm_report_client::set_report_verbosity_level` (see page 876).

function action `get_action(severity s, string id)`
This method looks up the action associated with this severity and id.

function FILE `get_file_handle(severity s, string id)`
This method looks up the file descriptor associated with reports with this severity and id.


```
function void report(severity s, string name, string id,  
                   string mess,  
                   int verbosity_level=0,  
                   avm_report_client client=null)
```

This is the basic reporting method, which is called by the four core reporting methods `avm_report_client::avm_report_message` (see page 875), `avm_report_client::avm_report_warning` (see page 875), `avm_report_client::avm_report_error` (see page 874), and `avm_report_client::avm_report_fatal` (see page 874). See the descriptions of these methods for their detailed behavior.

```
function string format_action(action a)
```

This method returns a string that describes the action.

```
function void set_severity_action(severity s, action a)
```

See `avm_report_client::set_report_severity_action` (see page 876).

```
function void set_id_action(string id, action a)
```

See `avm_report_client::set_report_id_action` (see page 876).

```
function void set_severity_id_action(severity s, string id,  
                                    action a)
```

See `avm_report_client::set_report_severity_id_action` (see page 877).

```
function void set_default_file(FILE f)
```

See `avm_report_client::set_report_default_file` (see page 877).

```
function void set_severity_file(severity s, FILE f)
```

See `avm_report_client::set_report_severity_file` (see 877).

```
function void set_id_file(string id, FILE f)
```

See `avm_report_client::set_report_id_file` (see page 877).

```
function void set_severity_id_file(severity s, string id,  
                                  FILE f)
```

See `avm_report_client::set_report_severity_id_file` (see page 877).

```
function void dump_state()
```

See `avm_report_client::dump_report_state` (see page 877).

avm_report_server

`avm_report_server` is a global server that processes all the reports generated by an `avm_report_handler`. None of its methods are intended to be called by normal test bench code, although in some circumstances the virtual methods `process_report` and/or `compose_message` may be overloaded in a subclass.

file

`reporting/avm_report_server.svh`

virtual

no

members

`static avm_report_server global_report_server=null`

This is an internal `avm_report_server` singleton.

`local int max_quit_count`

This specifies the maximum number of COUNT actions that can be tolerated before a COUNT action is treated as an EXIT action. The default value is 0, which is treated as specifying no upper bound.

`local int quit_count`

This is the actual number of COUNT actions sent to the server.

`local int severity_count[severity]`

This counts the number of messages for each severity.

`local int id_count[string]`

This counts the number of messages for each string id.

methods

`function new()`

The constructor is protected to enforce a singleton.

`static function avm_report_server get_server()`

This method returns a handle to the singleton.

`function int get_max_quit_count()`

This method gets the value of `max_quit_count()`.

`function void set_max_quit_count(int m)`

This method sets the value of `max_quit_count()`.

`function void reset_quit_count()`

This method resets the value of `quit_count` to 0.

`function void incr_quit_count()`

This method increments the value of `quit_count`.

function int get_quit_count()

This method gets the value of `quit_count`.

function bit is_quit_count_reached()

This method returns 1 if the value of `quit_count` has reached its upper bound, if there is one, and returns 0 otherwise.

function void reset_severity_counts()

This method resets the values in the `severity_count` array.

function int get_severity_count(severity s)

This method gets the number of reports with severity `s` since the last reset.

function void incr_severity_count(severity s)

This method increments the severity count for this severity.

function void set_id_count(string id, int n)

This method resets the value in the `id_count` array for an `id` to `n`.

function int get_id_count(string id)

This method gets the number of reports with this `id`.

function void incr_id_count(string id)

This method increments the number of reports with this `id`.

function void summarize(FILE f=0)

See `avm_report_client::report_summarize` (see [876](#)).

function void f_display(FILE f, string s)

This method sends string `s` to the command line if `f` is 0 and to the file(s) specified by `f` if it is not 0.

function void dump_server_state()

See `avm_report_client::dump_report_state()` (see [page 877](#)).

```
virtual function void process_report( severity s, string name ,
                                   string id, string message,
                                   action a,
                                   FILE f,
                                   string filename , int line,
                                   avm_report_client client )
```

This method calls `compose_message` to construct the actual message to be output. It then takes the appropriate action according to the value of `action a` and file `f`. This method can be overloaded by expert users so that the report system processes the actions different from the way described in `avm_report_client` and `avm_report_handler`.

```
virtual function string compose_message(severity s, string name,
                                       string id, string message)
```

This method constructs the actual string sent to the file or command line from the severity, component name, report id, and the message itself. Expert users can overload this method to change the formatting of the reports generated by `avm_report_client`.

avm_reporter

extends `avm_report_client`

`avm_reporter` is a reporter that can be used by objects that are not `avm_named_components` to issue reports.

file

`reporting/avm_report_client.svh`

virtual

no

members

<none>

methods

```
function new(string name="reporter")  
    The constructor has a default name of "reporter."
```

— Symbols —

, 322
 \$finish behavior, customizing, 707
 +typdelays, 635, 672
 {}, 21
 'hasX, hasX, 36

— Numerics —

0-In tools
 invoking from vcom, 521
 invoking from vlog, 614
 invoking from vopt, 644
 invoking from vsim, 678
 2001, keywords, disabling, 635

— A —

abort command, 58
 absolute time, using @, 28
 add atv command, 59
 add button command, 60
 add dataflow command, 63, 72
 add list command, 65
 add log command, 322
 add memory command, 70
 add watch command, 75
 add wave command, 76
 add_cmdhelp command, 83
 add_menu command, 85
 add_menucb command, 87
 add_menuitem simulator command, 89
 add_separator command, 91
 add_submenu command, 92
 addTime command, 467
 alias command, 94
 analog
 signal formatting, 77
 analysis_fifo class, 831
 analysis_if class, 839
 analysis_imp class, 826
 analysis_port class, 827

annotating interconnect delays,
 v2k_int_delays, 711
 archives, library, 611
 argument, 627
 arrays
 indexes, 17
 slices, 17, 21
 arrays, VHDL, searching for, 32
 assertion active command, 97
 assertion count command, 99
 assertion fail command, 104
 assertion pass command, 106
 assertion profile command, 108
 assertions
 count, 99
 enabling, 104, 106
 failure behavior, 104
 pass behavior, 106
 profiling, 108
 testing for with onbreak command, 353
 thread viewing, 59, 110
 assume directives
 -assume argument, 679
 -noassume argument, 688
 attributes, of signals, using in expressions, 36
 atv log command, 110
 avm_*_export class
 definition of, 800
 avm_*_imp class, 802
 avm_*_port class
 definition of, 804
 avm_algorithmic_comparator class, 793
 avm_analysis_port class, 806
 avm_blocking_master_imp class, 807
 avm_blocking_slave_imp class, 809
 avm_built_in_clone class, 864
 avm_built_in_comp class, 865
 avm_built_in_converter class, 866
 avm_built_in_pair class, 867
 avm_class_clone class, 868

avm_class_comp class, 869
 avm_class_converter class, 870
 avm_class_pair class, 871
 avm_connector_base class, 811
 avm_in_order_built_in_comparator class, 795
 avm_in_order_class_comparator class, 796
 avm_in_order_comparator class, 797
 avm_master_imp class, 815
 avm_named_component class
 definition of, 781
 avm_nonblocking_master_imp class, 817
 avm_nonblocking_slave_imp class, 819
 avm_port_base class, 821
 avm_random_stimulus class, 786
 avm_report_client class
 definition of, 874
 avm_report_handler class
 definition of, 879
 avm_report_server class, 882
 avm_reporter class, 884
 avm_slave_imp class, 823
 avm_stimulus class, 788
 avm_subscriber class, 789
 avm_threaded_component class
 definition of, 790
 avm_transaction class, 872
 avm_transport_imp class, 825
 avm_verification_component class, 791

— B —

batch_mode command, 112
 batch-mode simulations
 halting, 742
 bd (breakpoint delete) command, 113
 binary radix, mapping to std_logic values, 41
 blocking interfaces
 tlm_transport_if, 863
 bookmark add wave command, 115
 bookmark delete wave command, 117
 bookmark goto wave command, 118
 bookmark list wave command, 119
 bp (breakpoint) command, 120
 brackets, escaping, 21
 break
 on signal value, 737
 breakpoints

 conditional, 737
 continuing simulation after, 423
 deleting, 113
 listing, 120
 setting, 120
 signal breakpoints (when statements), 737
 time-based
 in when statements, 743
 built-in types
 cloning, 864
 comparing, 865
 converting to strings, 866
 bus contention checking, 135
 configuring, 137
 disabling, 138
 bus float checking
 configuring, 141
 disabling, 142
 enabling, 139
 busses
 escape characters in, 21
 user-defined, 81
 buttons, adding to the Main window toolbar, 60

— C —

C callstack
 moving down, 398
 moving up, 362
 C debugging, 128
 case choice, must be locally static, 529
 case sensitivity
 VHDL vs. Verilog, 21
 cd (change directory) command, 127
 cdbg command, 128
 change command, 131
 change_menu_cmd command, 134
 channel class, 831
 definitions of, 829
 tlm_fifo, 832
 tlm_req_rsp_channel, 835
 tlm_transport_channel, 838
 check contention add command, 135
 check contention config command, 137
 check contention off command, 138
 check float add command, 139
 check float config command, 141

- check float off command, [142](#)
- check stable off command, [143](#)
- check stable on command, [144](#)
- check_synthesis argument, [522](#)
- checkpoint command, [146](#)
- class member selection, syntax, [17](#)
- classes
 - cloning, [868](#)
 - comparing, [869](#)
 - converting to strings, [870](#)
- Code Coverage
 - coverage analyze command, [191](#)
 - coverage clear command, [201](#)
 - coverage exclude command, [205](#)
 - coverage goal command, [212](#)
 - coverage report command, [221](#)
 - coverage save command, [232](#)
 - coverage tag command, [234](#)
 - coverage testnames command, [238](#)
 - coverage unlinked command, [239](#)
 - coverage weight command, [242](#)
 - merging reports, [542](#)
 - toggle coverage
 - excluding signals, [473](#)
 - vcover report command, [554](#)
- Color
 - radix, [406](#)
 - example, [407](#)
- combining signals, busses, [81](#)
- command line args, accessing
 - vopt sc_arg command, [669](#)
 - vsim sc_arg command, [712](#)
- Commands
 - wlfman, [749](#)
- commands
 - .main clear, [57](#)
 - abort, [58](#)
 - add atv, [59](#)
 - add button, [60](#)
 - add dataflow, [63](#), [72](#)
 - add list, [65](#)
 - add memory, [70](#)
 - add testbrowser, [74](#)
 - add watch, [75](#)
 - add wave, [76](#)
 - add_menu, [85](#)
 - add_menucb, [87](#)
 - add_menuitem, [89](#)
 - add_separator, [91](#)
 - add_submenu, [92](#)
 - alias, [94](#)
 - assertion active, [97](#)
 - assertion count, [99](#)
 - assertion fail command, [104](#)
 - assertion pass, [106](#)
 - atv log, [110](#)
 - batch_mode, [112](#)
 - bd (breakpoint delete), [113](#)
 - bookmark add wave, [115](#)
 - bookmark delete wave, [117](#)
 - bookmark goto wave, [118](#)
 - bookmark list wave, [119](#)
 - bp (breakpoint), [120](#)
 - cd (change directory), [127](#)
 - cdbg, [128](#)
 - change, [131](#)
 - change_menu_cmd, [134](#)
 - check contention add, [135](#)
 - check contention config, [137](#)
 - check contention off, [138](#)
 - check float add, [139](#)
 - check float config, [141](#)
 - check float off, [142](#)
 - check stable off, [143](#)
 - check stable on, [144](#)
 - checkpoint, [146](#)
 - compare add, [148](#)
 - compare annotate, [154](#), [158](#)
 - compare clock, [156](#)
 - compare close, [162](#)
 - compare delete, [161](#)
 - compare info, [163](#)
 - compare list, [165](#)
 - compare open, [178](#)
 - compare options, [166](#)
 - compare reload, [171](#)
 - compare reset, [172](#)
 - compare run, [173](#)
 - compare savediffs, [174](#)
 - compare saverules, [175](#)

compare see, 176
 compare start, 173
 configure, 182
 context, 188
 coverage analyze, 191
 coverage attribute, 198
 coverage clear, 201
 coverage edit, 203
 coverage exclude, 205
 coverage goal, 212
 coverage open, 215
 coverage ranktest, 216
 coverage report, 221
 coverage save, 232
 coverage tag, 234
 coverage testnames, 238
 coverage unlinked, 239
 coverage weight, 242
 dataset alias, 245
 dataset clear, 246
 dataset close, 247
 dataset config, 248, 250
 dataset info, 251
 dataset list, 252
 dataset open, 253
 dataset rename, 254, 256
 dataset restart, 255
 dataset snapshot, 257
 delete, 260
 describe, 261
 disable_menu, 263
 disable_menuitem, 264
 disablebp, 262
 do, 265
 down, 267
 drivers, 270
 dumplog64, 271
 echo, 272
 edit, 273
 enable_menu, 275
 enable_menuitem, 276
 enablebp, 274
 environment, 278
 examine, 280
 exit, 285

fcover configure, 286
 find, 289
 force, 303
 fsm list, 307, 309
 fsm properties, 308
 getactivecursortime, 311
 getactivemarkertime, 312
 layout, 316
 lecho, 318
 left, 319
 log, 322
 lshift, 326
 lsublist, 327
 mem compare, 328
 mem display, 329
 mem list, 332
 mem load, 333
 mem save, 337
 mem search, 340
 messages clearfilter, 343, 344
 next, 346
 noforce, 347
 nolog, 348
 notation conventions, 15
 notepad, 350
 noview, 351
 nowhen, 352
 onbreak, 353
 onElabError, 355
 onerror, 356
 pa msg, 359
 pause, 361
 pop, 362
 power add, 363
 power report, 369
 power reset, 372
 printenv, 374, 375
 process report, 376
 profile clear, 377
 profile interval, 378
 profile off, 379
 profile on, 380
 profile open, 382
 profile option, 383
 profile reload, 385

profile report, 386
 profile save, 390
 property list, 394
 property wave, 396
 push, 398
 pwd, 399
 quietly, 400
 quit, 401
 qverilog, 402
 radix, 404
 radix define, 406
 radix list, 410
 radix name, 411
 readers, 413
 report, 414
 restart, 416
 restore, 418
 resume, 419
 right, 420
 run, 423
 runStatus, 426
 sccom, 428
 scgenmod, 436
 sdfcom, 439
 search, 441
 searchlog, 444
 see, 447
 seetime, 448
 setenv, 450
 shift, 451
 show, 452
 simstats, 453
 status, 455
 stop, 457
 suppress, 458
 tb (traceback), 459
 tcheck_set, 460
 tcheck_status, 464
 Time, 467
 toggle add, 470
 toggle disable, 473
 toggle enable, 474
 toggle report, 475
 toggle reset, 477
 tr color, 478
 tr id, 483
 tr order, 481
 transcribe, 485
 transcript, 486
 transcript file, 487
 transcript path, 489
 transcript sizelimit, 490
 TreeUpdate, 756
 tssi2mti, 491
 typespec, 492
 unsetenv, 495
 up, 496
 variables referenced in, 28
 vcd add, 498
 vcd checkpoint, 500
 vcd comment, 501
 vcd dumpports, 502
 vcd dumpportsall, 505
 vcd dumpportsflush, 506
 vcd dumpportslimit, 507
 vcd dumpportsoff, 508
 vcd dumpportson, 509
 vcd file, 510
 vcd files, 512
 vcd flush, 514
 vcd limit, 515
 vcd off, 516
 vcd on, 517
 vcom, 520
 vcover attribute, 538
 vcover dump, 540
 vcover merge, 542
 vcover ranktest, 548
 vcover report, 554
 vcover testnames, 568
 vdel, 571
 vdir, 573
 vencrypt, 576
 verror, 578
 vgencomp, 580
 vhencrypt, 582
 view, 584
 virtual count, 588
 virtual define, 589
 virtual delete, 590

- virtual describe, 591
- virtual expand, 592
- virtual function, 593
- virtual hide, 596
- virtual log, 597
- virtual nohide, 599
- virtual nolog, 600
- virtual region, 602
- virtual save, 603
- virtual show, 604
- virtual signal, 605
- virtual type, 609
- vlib, 611
- vlog, 613
- vmake, 639
- vmap, 641
- vopt, 642
- vsim, 676
- vsimDate, 715
- vsimId, 715
- vsimVersion, 715
- vsourc, 717
- wave, 718
- wave create, 722
- wave edit, 728
- wave export, 731
- wave import, 732
- wave modify, 733
- wave sort, 736
- WaveActivateNextPane, 756
- WaveRestoreCursors, 756
- WaveRestoreZoom, 756
- when, 737
- where, 744
- wlf2log, 745
- wlf2vcd, 748
- wlfrecover, 753
- write cell_report, 754
- write format, 755
- write list, 757
- write preferences, 758
- write report, 759
- write timing, 762
- write transcript, 763
- write tssi, 764
- write wave, 766
- xml2ucdb, 768
- comment characters in VSIM commands, 15
- comparator class
 - avm_in_order_built_in_comparator, 795
 - avm_in_order_class_comparator, 796
 - avm_in_order_comparator, 797
 - definitions for, 791
 - tlm_algorithmic_comparator, 793
- compare add command, 148
- compare annotate command, 154, 158
- compare clock command, 156
- compare close command, 162
- compare delete command, 161
- compare info command, 163
- compare list command, 165
- compare open command, 178
- compare options command, 166
- compare reload command, 171
- compare reset command, 172
- compare run command, 173
- compare savediffs command, 174
- compare saverules command, 175
- compare see command, 176
- compare start command, 173
- compatibility, of vendor libraries, 573
- compiling
 - range checking in VHDL, 534
 - SystemC, 428, 436
 - Verilog, 613
 - VHDL, 520
 - at a specified line number, 528
 - selected design units (-just eapbc), 527
 - standard package (-s), 534, 634
- component classes
 - avm_named_component, 781
 - avm_random_stimulus, 786
 - avm_stimulus, 788
 - avm_subscriber, 789
 - avm_threaded_component, 790
 - avm_verification_component, 791
 - general description of, 778
- compressing files
 - checkpoint files, 146
 - elaboration files, 680

VCD files, [502](#), [512](#)

concatenation

- directives, [40](#)
- of signals, [39](#)

conditional breakpoints, [737](#)

configurations, simulating, [676](#)

configure command, [182](#)

connector classes

- analysis_imp, [826](#)
- analysis_port, [827](#)
- avm_*_export, [800](#)
- avm_*_imp, [802](#)
- avm_*_port, [804](#)
- avm_analysis_port, [806](#)
- avm_blocking_master_imp, [807](#)
- avm_blocking_slave_imp, [809](#)
- avm_connector_base, [811](#)
- avm_master_imp, [815](#)
- avm_nonblocking_master_imp, [817](#)
- avm_nonblocking_slave_imp, [819](#)
- avm_port_base, [821](#)
- avm_slave_imp, [823](#)
- avm_transport_imp, [825](#)
- deprecated implementations, list of, [829](#)
- general description of, [799](#)
- global_analysis_ports, [828](#)

constants

- in case statements, [529](#)
- values of, displaying, [261](#), [280](#)

contention checking, [135](#)

context command, [188](#)

conversion

- radix, [404](#)

coverage

- vcover testnames command, [568](#)
- coverage analyze command, [191](#)
- coverage attribute command, [198](#)
- coverage clear command, [201](#)
- coverage edit command, [203](#)
- coverage exclude command, [205](#)
- coverage goal command, [212](#)
- coverage open command, [215](#)
- coverage ranktest command, [216](#)
- coverage report command, [221](#)
- coverage save command, [232](#)

- coverage tag command, [234](#)
- coverage testnames command, [238](#)
- coverage unlinked command, [239](#)
- Coverage View mode
 - coverage open command, [215](#)
- coverage weight command, [242](#)
- customizing
 - adding buttons, [60](#)

— D —

dataset alias command, [245](#)

dataset clear command, [246](#)

dataset close command, [247](#)

dataset config command, [248](#), [250](#)

dataset info command, [251](#)

dataset list command, [252](#)

dataset open command, [253](#)

dataset rename command, [254](#), [256](#)

dataset restart command, [255](#)

dataset snapshot command, [257](#)

datasets

- environment command, specifying with, [278](#)

declarations, hiding implicit with explicit, [537](#)

default VOPT behavior, and .ini file, [642](#)

+define+, [619](#)

delay

- interconnect, [688](#)

+delay_mode_distributed, [619](#), [649](#)

+delay_mode_path, [619](#), [650](#)

+delay_mode_unit, [619](#), [650](#)

+delay_mode_zero, [620](#), [650](#)

'delayed, [36](#)

delete command, [260](#)

deltas

- collapsing in WLF files, [699](#)

dependencies, checking, [573](#)

dependency errors, [527](#), [621](#)

describe command, [261](#)

design loading, interrupting, [676](#)

design units

- report of units simulated, [759](#)

Verilog

- adding to a library, [613](#)

directories

- mapping libraries, [641](#)

disable_menu command, 263
 disable_menuitem command, 264
 disablebp command, 262
 dividers
 adding from command line, 77
 divTime ccommand, 467
 do command, 265
 DO files (macros), 265
 down command, 267
 -dpiheader, vlog, 620
 -dpiheader, vopt, 650
 drivers command, 270
 dump files, viewing in the simulator, 518
 dumplog64 command, 271

— E —

echo command, 272
 edges, finding, 319, 420
 edit command, 273
 enable_menu command, 275
 enable_menuitem command, 276
 enablebp command, 274
 encryption
 +protect argument, 632
 -nodebug argument (vcom), 530
 -nodebug argument (vlog), 628
 entities, specifying for simulation, 713
 enumerated types
 user defined, 609
 environment command, 278
 environment variables
 reading into Verilog code, 619
 specifying UNIX editor, 273
 state of, 375
 using in pathnames, 21
 environment, displaying or changing
 pathname, 278
 eqTime command, 467
 errors
 getting details about messages, 578
 onerror command, 356
 SDF, disabling, 694
 escape character, 21
 event order
 changing in Verilog, 616, 619, 646
 examine command, 280

exit command, 285
 exiting the simulator, customizing behavior,
 707
 extended identifier, 35
 extended identifiers, 22
 extended toggle coverage, reporting, 224, 555,
 556

— F —

-f, 620, 650
 fcover configue command, 286
 FIFO channel
 AVM class for, 829
 file compression
 checkpoint files, 146
 elaboration files, 680
 VCD files, 502, 512
 find command, 289
 force command, 303
 foreign module declaration
 Verilog example, 438
 format file
 List window, 755
 Wave window, 755
 formatTime command, 468
 fsm list command, 307, 309
 fsm properties command, 308
 Functional coverage
 merging databases offline, 542
 functional coverage
 configuring directives, 286

— G —

generics
 assigning or overriding values with -g and -
 G, 683
 examining generic values, 280
 limitation on assigning composite types,
 652, 683
 getactivecursortime command, 311
 getactivemarkertime command, 312
 glitches
 disabling generation
 from command line, 702
 global visibility
 PLI/FLI shared objects, 684

global_analysis_ports class, 828

gteTime command, 467

gtTime command, 467

GUI_expression_format, 32

 syntax, 33

— H —

'hasX, 36

hazards

 -hazards argument to vlog, 622, 653

 -hazards argument to vsim, 704

history

 of commands

 shortcuts for reuse, 30

HTML report

 generating from .ucdb, 225, 558

— | —

implementation class, related interfaces, 802

implicit operator, hiding with vcom -explicit,
537

+incdir+, 623

indexed arrays, escaping square brackets, 21

interconnect delays, 688

 annotating per Verilog 2001, 711

interfaces

 related implementation, 802

internal signals, adding to a VCD file, 498

interrupting design loading, 676

intToTime command, 467

— K —

keywords

 disabling 2001 keywords, 635

 enabling SystemVerilog keywords, 634

— L —

layout command, 316

LD_LIBRARY_PATH, disabling default

 internal setting of, 688

lecho command, 318

left command, 319

+libcell, 625

libraries

 archives, 611

 dependencies, checking, 573

 design libraries, creating, 611

 listing contents, 573

 refreshing library images, 534, 633

 vendor supplied, compatibility of, 573

 Verilog, 685

lint-style checks, 626

List window

 adding items to, 65

lm, 852

loading designs, interrupting, 676

Local variables

 print to Transcript, 105

log command, 322

log file

 log command, 322

 nolog command, 348

 QuickSim II format, 745

 redirecting with -l, 685, 686

 virtual log command, 597

 virtual nolog command, 600

Log local variables, 105

lshift command, 326

lsublist command, 327

lteTime command, 467

ltTime command, 467

— M —

macros (DO files)

 breakpoints, executing at, 122

 executing, 265

 forcing signals, nets, or registers, 303

 parameters

 passing, 265

 relative directories, 265

 shifting parameter values, 451

.main clear command, 57

master slave library (SystemC), including, 432

+maxdelays, 627, 657

mc_scan_plusargs, PLI routine, 708

mem compare command, 328

mem display command, 329

mem list command, 332

mem load command, 333

mem save command, 337

mem search command, 340

memory window

 add memory command, 70

adding items to, [70](#)
memory, comparing contents, [328](#)
memory, displaying contents, [329](#)
memory, listing, [332](#)
memory, loading contents, [333](#)
memory, saving contents, [337](#)
memory, searching for patterns, [340](#)
merge
 test-associated data, viewing, [191](#)
merging coverage reports, [542](#)
messages
 base class for, [874](#)
 echoing, [272](#)
 getting more information, [578](#)
 loading, disabling with -quiet, [633](#), [669](#)
 loading, disabling with -quiet, [534](#)
messages clearfilter command, [343](#), [344](#)
-mfcu, [627](#)
+mindelays, [627](#), [657](#)
mnemonics, assigning to signal values, [609](#)
modelsim.ini
 default VOPT behavior, [642](#)
mulTime command, [467](#)
multi-source interconnect delays, [688](#)

— N —

name case sensitivity, VHDL vs. Verilog, [21](#)
ncsim, one step simulation, [402](#)
negative pulses
 driving an error state, [709](#)
neqTime command, [467](#)
nets
 drivers of, displaying, [270](#)
 readers of, displaying, [413](#)
 stimulus, [303](#)
 values of
 examining, [280](#)
next command, [346](#)
-no_risefall_delaynets, [705](#)
-nodebug argument (vcom), [530](#)
-nodebug argument (vlog), [628](#)
noforce command, [347](#)
+nolibcell, [629](#), [658](#)
nolog command, [348](#)
nonblocking interfaces
 tlm_nonblocking_put_if, [856](#)

 tlm_nonblocking_slave_if, [857](#)
notepad command, [350](#)
noview command, [351](#)
+nowarn<CODE>, [631](#), [659](#)
nowhen command, [352](#)

— O —

object_list_file, WLF files, [749](#)
onbreak command, [353](#)
onElabError command, [355](#)
onerror command, [356](#)
optimizations
 disabling for Verilog designs, [631](#), [660](#)
 disabling for VHDL designs, [533](#)
 optimizing wlf files, [749](#)
 vopt command, [642](#)
optimizations (vlog)
 disabling process merging, [615](#)
order of events
 changing in Verilog, [616](#), [619](#), [646](#)

— P —

pa msg command, [359](#)
parameters
 using with macros, [265](#)
pathnames
 in VSIM commands, [16](#)
 spaces in, [16](#)
pause command, [361](#)
performance
 vopt command, [642](#)
PLI
 loading shared objects with global symbol
 visibility, [684](#)
policy class, transactions, [864](#)
pop command, [362](#)
power add command, [363](#)
Power Aware verification, [625](#), [661](#), [690](#)
power report command, [369](#)
power reset command, [372](#)
printenv command, [374](#), [375](#)
process report command, [376](#)
processes (vlog)
 optimizations, disabling the merging of,
 [615](#)
profile clear command, [377](#)

profile interval command, 378
 profile off command, 379
 profile on command, 380
 profile open command, 382
 profile option command, 383
 profile reload command, 385
 profile report command, 386
 profile save command, 390
 projects
 override mapping for work directory with
 vcom, 433, 536
 override mapping for work directory with
 vlog, 403, 636
 propagation, preventing X propagation, 689
 property list command, 394
 property wave command, 396
 pulse error state, 709
 push command, 398
 pwd command, 399

— Q —

QuickSim II logfile format, 745
 quietly command, 400
 quit command, 401
 qverilog command, 402

— R —

Radix
 color, 406
 example, 407
 user defined, 406
 radix
 character strings, displaying, 609
 display values in debug windows, 404
 of signals being examined, 67, 80, 282
 radix command, 404
 Radix define command, 406
 setting radix color, 406, 407
 radix list command, 410
 radix name command, 411
 range checking
 disabling, 532
 enabling, 534
 readers command, 413
 RealToTime command, 468
 record field selection, syntax, 17

refresh, dependency check errors, 527, 621
 refreshing library images, 534, 633
 report command, 414
 report handling, class for, 879
 report processing, global server for, 882
 reporting
 issuing reports, class for, 884
 processes in the Process Window, 376
 variable settings, 28
 reporting classes
 avm_report_client, 874
 avm_report_handler, 879
 avm_report_server, 882
 avm_reporter, 884
 general description of, 873
 resolution
 specifying with -t argument, 695
 restart command, 416
 restore command, 418
 resume command, 419
 right command, 420
 run command, 423
 runStatus command, 426

— S —

sc_stop()
 customizing simulator behavior, 707
 scaleTime command, 468
 sccom command, 428
 -scdpidebug command, 712
 scgenmod command, 436
 -sclib command, 669, 712
 scope resolution operator, 18
 scope, setting region environment, 278
 SCV library, including, 433
 SDF
 annotation verbose mode, 694
 compiled SDF, 439
 controlling missing instance messages,
 670, 694
 disabling individual checks, 460
 errors on loading, disabling, 694
 warning messages, disabling, 694
 sdfcom command, 439
 search command, 441
 search libraries, 685

- searching
 - binary signal values in the GUI, 41
 - List window
 - signal values, transitions, and names, 32, 267, 496
 - next and previous edge in Wave window, 319, 420
 - VHDL arrays, 32
 - Wave window
 - signal values, edges and names, 319, 420
- searchlog command, 444
- see command, 447
- seetime command, 448
- setenv command, 450
- shared objects
 - loading with global symbol visibility, 684
- shift command, 451
- shortcuts
 - command history, 30
 - command line caveat, 30
- show command, 452
- signals
 - alternative names in the Wave window (-label), 78
 - attributes of, using in expressions, 36
 - breakpoints, 737
 - combining into a user-defined bus, 81
 - drivers of, displaying, 270
 - environment of, displaying, 278
 - force time, specifying, 305
 - log file, creating, 322
 - pathnames in VSIM commands, 16
 - radix
 - specifying for examine, 67, 80, 282
 - readers of, displaying, 413
 - states of, displaying as mnemonics, 609
 - stimulus, 303
 - values of
 - examining, 280
 - replacing with text, 609
- simstats command, 453
- simulating
 - delays, specifying time units for, 28
 - design unit, specifying, 676
 - ncsim style, 402
 - one step, 402
 - saving simulations, 322, 698
 - stopping simulation in batch mode, 742
- simulations
 - saving results, 256, 257
- Simulator commands, 58
- simulator resolution
 - vsim -t argument, 695
- simulator version, 697, 715
- simultaneous events in Verilog
 - changing order, 616, 619, 646
- sml2ucdb command, 768
- source annotation, 432
- spaces in pathnames, 16
- sparse memories
 - listing with write report, 760
- specify path delays, 709
- square brackets, escaping, 21
- stability checking
 - disabling, 143
 - enabling, 144
- startup
 - alternate to startup.do (vsim -do), 681
- status command, 455
- Std_logic
 - mapping to binary radix, 41
- stop command, 457
- subTime command, 467
- suppress command, 458
- symbolic constants, displaying, 609
- symbolic names, assigning to signal values, 609
- synthesis
 - rule compliance checking, 522
- SystemC
 - class and structure member naming syntax, 17
 - DPI, command for single-stepping across call boundaries, 712
 - master slave library, including, 432
 - specifying shared library path, command, 669, 712
 - verification library, including, 433
- SystemVerilog

- enabling with -sv argument, [634](#)
- multiple files in a compilation unit, [627](#)
- scope resolution, [18](#)
- SystemVerilog assertions
 - disabling, [689](#)
- T —
- tb command, [459](#)
- tcheck_set command, [460](#)
- tcheck_status command, [464](#)
- Tcl
 - history shortcuts, [30](#)
 - variable
 - in when commands, [740](#)
- test management window
 - adding UCDB files to, [74](#)
- test-associated data
 - viewing, [191](#)
- TFMPC
 - disabling warning, [707](#)
- time
 - absolute, using @, [28](#)
 - simulation time units, [28](#)
- time collapsing, [699](#)
- Time commands, [467](#)
- time resolution
 - setting
 - with vsim command, [695](#)
- time, time units, simulation time, [28](#)
- timescale directive warning
 - disabling, [707](#)
- timing
 - disabling checks, [630](#), [659](#)
 - disabling checks for entire design, [690](#)
 - disabling individual checks, [460](#)
 - status of individual checks, [464](#)
- title, Main window, changing, [696](#)
- TLM interface classes
 - analysis_if, [839](#)
 - general description of, [838](#)
 - tlm_blocking_get_if, [840](#)
 - tlm_blocking_get_peek_if, [841](#)
 - tlm_blocking_master_if, [844](#)
 - tlm_blocking_put_if, [843](#)
 - tlm_blocking_slave_if, [845](#)
 - tlm_get_if, [846](#)
 - tlm_get_peek_if, [847](#)
 - tlm_master_if, [849](#)
 - tlm_nonblocking_get_peek_if, [852](#)
 - tlm_nonblocking_get_if, [851](#)
 - tlm_nonblocking_master_if, [853](#)
 - tlm_nonblocking_peek_if, [855](#)
 - tlm_nonblocking_put_if, [856](#)
 - tlm_nonblocking_slave_if, [857](#)
 - tlm_peek_if, [859](#)
 - tlm_put_if, [860](#)
 - tlm_slave_if, [861](#)
 - tlm_transport_if, [863](#)
 - tlm_*_imp, deprecated implementations, [829](#)
 - tlm_blocking_get_if class, [840](#)
 - tlm_blocking_get_peek_if class, [841](#)
 - tlm_blocking_master_if class, [844](#)
 - tlm_blocking_put_if class, [843](#)
 - tlm_blocking_slave_if class, [845](#)
 - tlm_fifo class, [832](#)
 - tlm_get_if class, [846](#), [847](#)
 - tlm_master_if class, [849](#)
 - tlm_nonblocking_get_if class, [851](#)
 - tlm_nonblocking_get_peek_if class, [852](#)
 - tlm_nonblocking_master_if class, [853](#)
 - tlm_nonblocking_peek_if class, [855](#)
 - tlm_nonblocking_put_if class, [856](#)
 - tlm_nonblocking_slave_if class, [857](#)
 - tlm_peek_if class, [859](#)
 - tlm_put_if class, [860](#)
 - tlm_req_rsp_channel class, [835](#)
 - tlm_slave_if class, [861](#)
 - tlm_transport_channel class, [838](#)
 - tlm_transport_if class, [863](#)
- toggle
 - reporting extended, [224](#), [555](#), [556](#)
- toggle add command, [470](#)
- toggle coverage
 - excluding signals, [473](#)
 - reporting, duplication of elements, [475](#)
 - reporting, ordering of nodes, [475](#)
- toggle disable command, [473](#)
- toggle enable command, [474](#)
- toggle report command, [475](#)
- toggle reset command, [477](#)
- toggle statistics

enabling, 470
 reporting, 475
 resetting, 477
 tr color command, 478
 tr id command, 483
 tr order command, 481
 transactions policy classes
 AVM transactions, base class for, 872
 avm_built_in_clone, 864
 avm_built_in_comp, 865
 avm_built_in_converter, 866
 avm_built_in_pair, 867
 avm_class_clone, 868
 avm_class_comp, 869
 avm_class_converter, 870
 avm_class_pair, 871
 general description of, 863
 transcribe command, 485
 transcript
 clearing, 57
 redirecting with -l, 685, 686
 transcript command, 486
 transcript file command, 487
 transcript path command, 489
 transcript sizelimit command, 490
 transitions, signal, finding, 319, 420
 TreeUpdate command, 756
 TSCALE, disabling warning, 707
 TSSI, 764
 tssi2mti command, 491
 typespec command, 492

— U —

-u, 635
 UCDB coverage
 coverage attribute command, 198
 vcover attribute command, 538
 vcover dump command, 540
 undeclared nets, reporting an error, 626
 Unified Power Format, 668
 unsetenv command, 495
 up command, 496
 UPF, 625
 UPF (Unified Power Format), 668
 user-defined bus, 81
 User-defined radix, 406

— V —

-v, 635
 v2k_int_delays, 711
 validTime command, 468
 values
 describe HDL items, 261
 examine HDL item values, 280
 replacing signal values with strings, 609
 variable settings report, 28
 variables
 describing, 261
 referencing in commands, 28
 value of
 changing from command line, 131
 examining, 280
 vcd add command, 498
 vcd checkpoint command, 500
 vcd comment command, 501
 vcd dumpports command, 502
 vcd dumpportsall command, 505
 vcd dumpportsflush command, 506
 vcd dumpportslimit command, 507
 vcd dumpportsoff command, 508
 vcd dumpportson command, 509
 vcd file command, 510
 VCD files
 adding items to the file, 498
 capturing port driver data, 502
 converting to WLF files, 518
 creating, 498
 dumping variable values, 500
 flushing the buffer contents, 514
 generating from WLF files, 748
 inserting comments, 501
 internal signals, adding, 498
 specifying maximum file size, 515
 specifying name of, 512
 specifying the file name, 510
 state mapping, 510, 512
 turn off VCD dumping, 516
 turn on VCD dumping, 517
 viewing files from another tool, 518
 vcd files command, 512
 vcd flush command, 514
 vcd limit command, 515

- vcd off command, [516](#)
 - vcd on command, [517](#)
 - vcd2wlf command, [518](#)
 - vcom command, [520](#)
 - vcom Examples, [536](#)
 - vcover attribute command, [538](#)
 - vcover dump command, [540](#)
 - vcover merge command, [542](#)
 - vcover ranktest command, [548](#)
 - vcover report command, [554](#)
 - vcover testnames command, [568](#)
 - vdel command, [571](#)
 - vdir command, [573](#)
 - vector elements, initializing, [131](#)
 - vencrypt command, [576](#)
 - vendor libraries, compatibility of, [573](#)
 - Verilog
 - \$finish behavior, customizing, [707](#)
 - capturing port driver data with -dumpports, [510](#)
 - Verilog 2001
 - disabling support, [635](#)
 - verror command, [578](#)
 - version
 - obtaining with vsim command, [697](#)
 - obtaining with vsim<info> commands, [715](#)
 - vgencomp command, [580](#)
 - VHDL
 - arrays
 - searching for, [32](#)
 - compile
 - 1076-2008, [521](#)
 - conditions and expressions, automatic conversion of H and L., [529](#)
 - field naming syntax, [17](#)
 - VHDL-1993, enabling support for, [521](#)
 - VHDL-2002, enabling support for, [521](#)
 - vhencrypt command, [582](#)
 - view command, [584](#)
 - viewing
 - waveforms, [698](#)
 - virtual count commands, [588](#)
 - virtual define command, [589](#)
 - virtual delete command, [590](#)
 - virtual describe command, [591](#)
 - virtual expand commands, [592](#)
 - virtual function command, [593](#)
 - virtual hide command, [596](#)
 - virtual log command, [597](#)
 - virtual nohide command, [599](#)
 - virtual nolog command, [600](#)
 - virtual region command, [602](#)
 - virtual save command, [603](#)
 - virtual show command, [604](#)
 - virtual signal command, [605](#)
 - virtual type command, [609](#)
 - vlib command, [611](#)
 - vlog
 - multiple file compilation, [627](#)
 - vlog command, [613](#)
 - vmake command, [639](#)
 - vmap command, [641](#)
 - vopt
 - path separator, [642](#)
 - vopt command, [642](#)
 - vsim
 - disabling internal setting of LD_LIBRARY_PATH, [688](#)
 - vsim build date and version, [715](#)
 - vsim command, [676](#)
 - vsim Examples, [713](#)
- W —**
- WARNING[8], -lint argument to vlog, [626](#)
 - warnings
 - SDF, disabling, [694](#)
 - suppressing VCOM warning messages, [532](#), [631](#), [659](#)
 - suppressing VLOG warning messages, [631](#), [659](#)
 - suppressing VSIM warning messages, [707](#)
 - watch window
 - add watch command, [75](#)
 - adding items to, [75](#)
 - watching signal values, [75](#)
 - wave commands, [718](#)
 - wave create command, [722](#)
 - wave cursor commands, [718](#)
 - wave edit command, [728](#)
 - wave export command, [731](#)
 - wave import command, [732](#)

- wave log format (WLF) file, 698
 - wave modify command, 733
 - wave sort command, 736
 - Wave window
 - adding items to, 76
 - WaveActivateNextPane command, 756
 - Waveform Comparison, 148
 - waveform editor
 - creating waves, 722
 - editing commands, 728
 - importing vcd stimulus file, 732
 - modifying existing waves, 733
 - saving waves, 731
 - waveform logfile
 - log command, 322
 - waveforms
 - optimizing viewing of, 700
 - saving and viewing, 322
 - WaveRestoreCursors command, 756
 - WaveRestoreZoom command, 756
 - when command, 737
 - when statement
 - time-based breakpoints, 743
 - where command, 744
 - wildcard characters
 - for pattern matching in simulator commands, 22
 - windows
 - List window
 - output file, 757
 - saving the format of, 755
 - Main window
 - adding user-defined buttons, 60
 - opening
 - from command line, 584
 - Wave window
 - path elements, changing, 186
 - WLF files
 - collapsing deltas, 699
 - collapsing time steps, 699
 - converting to VCD, 748
 - creating from VCD, 518
 - indexing, 749
 - limiting size, 700
 - log command, 322
 - merging, 749
 - optimizing, 749
 - optimizing waveform viewing, 700
 - repairing, 753
 - saving, 256, 257
 - specifying name, 698
 - wlfman command, 749
 - wlf2log command, 745
 - wlf2vcd command, 748
 - wlfman command, 749
 - wlfrecover command, 753
 - write cell_report command, 754
 - write format command, 755
 - write list command, 757
 - write preferences command, 758
 - write report command, 759
 - write timing command, 762
 - write transcript command, 763
 - write tssi command, 764
 - write wave command, 766
- X —
- X propagation
 - disabling for entire design, 689
 - disabling X generation on specific instances, 460
- Y —
- y, 636
- Z —
- zoom
 - wave window
 - returning current range, 719

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE SOFTWARE. USE OF SOFTWARE INDICATES YOUR COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT (“Agreement”)

This is a legal agreement concerning the use of Software (as defined in Section 2) between the company acquiring the license (“Customer”), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity (“Mentor Graphics”). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, if received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. ORDERS, FEES AND PAYMENT.

- 1.1. To the extent Customer (or if and as agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (“Order(s)”), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.
- 1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will invoice separately. Unless provided with a certificate of exemption, Mentor Graphics will invoice Customer for all applicable taxes. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. Notwithstanding anything to the contrary, if Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under such orders in the event of default by the third party.
- 1.3. All products are delivered FCA factory (Incoterms 2000) except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all products delivered under this Agreement, to secure payment of the purchase price of such products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data (“Software”) are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form; (b) for Customer's internal business purposes; (c) for the term; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of receiving support or consulting services, evaluating Software or

otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.
4. **BETA CODE.**
 - 4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.
 - 4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.
 - 4.3. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.
5. **RESTRICTIONS ON USE.**
 - 5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Software available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Software and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Log files, data files, rule files and script files generated by or for the Software (collectively "Files") constitute and/or include confidential information of Mentor Graphics. Customer may share Files with third parties excluding Mentor Graphics competitors provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") mean Mentor Graphics' proprietary syntaxes for expressing process rules. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or allow its use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive from Software any source code.
 - 5.2. Customer may not sublicense, assign or otherwise transfer Software, this Agreement or the rights under it, whether by operation of law or otherwise ("attempted transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable transfer charges. Any attempted transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.
 - 5.3. The provisions of this Section 5 shall survive the termination of this Agreement.
6. **SUPPORT SERVICES.** To the extent Customer purchases support services for Software, Mentor Graphics will provide Customer with available updates and technical support for the Software which are made generally available by Mentor Graphics as part of such services in accordance with Mentor Graphics' then current End-User Software Support Terms located at <http://supportnet.mentor.com/about/legal/>.

7. LIMITED WARRANTY.

7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Software will meet Customer's requirements or that operation of Software will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under the applicable Order and does not renew or reset, by way of example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF SOFTWARE TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF SOFTWARE THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) SOFTWARE WHICH IS LICENSED AT NO COST; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO SOFTWARE OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

8. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE SOFTWARE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

9. **LIFE ENDANGERING APPLICATIONS.** NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF SOFTWARE IN ANY APPLICATION WHERE THE FAILURE OR INACCURACY OF THE SOFTWARE MIGHT RESULT IN DEATH OR PERSONAL INJURY. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH CUSTOMER'S USE OF SOFTWARE AS DESCRIBED IN SECTION 9. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. INFRINGEMENT.

11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Software product infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay any costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense, (a) replace or modify Software so that it becomes noninfringing, or (b) procure for Customer the right to continue using Software, or (c) require the return of Software and refund to Customer any license fee paid, less a reasonable allowance for use.

11.3. Mentor Graphics has no liability to Customer if the claim is based upon: (a) the combination of Software with any product not furnished by Mentor Graphics; (b) the modification of Software other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of Software as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code; (g) any Software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

11.4. THIS SECTION IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.

12. **TERM.**

- 12.1. This Agreement remains effective until expiration or termination. This Agreement will immediately terminate upon notice if you exceed the scope of license granted or otherwise fail to comply with the provisions of Sections 2, 3, or 5. For any other material breach under this Agreement, Mentor Graphics may terminate this Agreement upon 30 days written notice if you are in material breach and fail to cure such breach within the 30 day notice period. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.
 - 12.2. Mentor Graphics may terminate this Agreement immediately upon notice in the event Customer is insolvent or subject to a petition for (a) the appointment of an administrator, receiver or similar appointee; or (b) winding up, dissolution or bankruptcy.
 - 12.3. Upon termination of this Agreement or any Software license under this Agreement, Customer shall ensure that all use of the affected Software ceases, and shall return it to Mentor Graphics or certify its deletion and destruction, including all copies, to Mentor Graphics' reasonable satisfaction.
 - 12.4. Termination of this Agreement or any Software license granted hereunder will not affect Customer's obligation to pay for products shipped or licenses granted prior to the termination, which amounts shall immediately be payable at the date of termination.
13. **EXPORT.** Software is subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct products of the products to certain countries and certain persons. Customer agrees that it will not export Software or a direct product of Software in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.
 14. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.
 15. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.
 16. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXIm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this section shall survive the termination of this Agreement.
 17. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of the Mentor Graphics intellectual property rights licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: This Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia (except for Japan) arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the Chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.
 18. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.
 19. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. All notices required or authorized under this Agreement must be in writing and shall be sent to the person who signs this Agreement, at the address specified below. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.