



A Siemens Business

ModelSim® Command Reference Manual

Software Version 2020.1

Unpublished work. © Siemens 2020

This document contains information that is confidential and proprietary to Mentor Graphics Corporation, Siemens Industry Software Inc., or their affiliates (collectively, "Siemens"). 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 confidential and proprietary information.

This document is for information and instruction purposes. Siemens 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 Siemens to determine whether any changes have been made.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. **End User License Agreement** — You can print a copy of the End User License Agreement from: mentor.com/eula.

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 Siemens whatsoever.

SIEMENS MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

SIEMENS SHALL NOT BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL OR PUNITIVE DAMAGES, LOST DATA OR PROFITS, EVEN IF SUCH DAMAGES WERE FORESEEABLE, ARISING OUT OF OR RELATED TO THIS PUBLICATION OR THE INFORMATION CONTAINED IN IT, EVEN IF SIEMENS HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

LICENSE RIGHTS APPLICABLE TO THE U.S. GOVERNMENT: This document explains the capabilities of commercial products that were developed exclusively at private expense. If the products are acquired directly or indirectly for use by the U.S. Government, then the parties agree that the products and this document are considered "Commercial Items" and "Commercial Computer Software" or "Computer Software Documentation," as defined in 48 C.F.R. §2.101 and 48 C.F.R. §252.227-7014(a)(1) and (a)(5), as applicable. Software and this document may only be used under the terms and conditions of the End User License Agreement referenced above as required by 48 C.F.R. §12.212 and 48 C.F.R. §227.7202. The U.S. Government will only have the rights set forth in the End User License Agreement, which supersedes any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.

TRADEMARKS: The trademarks, logos and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: www.plm.automation.siemens.com/global/en/legal/trademarks.html and mentor.com/trademarks.

The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Support Center: support.sw.siemens.com

Send Feedback on Documentation: support.sw.siemens.com/doc_feedback_form

Table of Contents

Chapter 1	
Syntax and Conventions.....	13
Documentation Conventions	13
File and Directory Pathnames	14
Design Object Names	15
Object Name Syntax	15
Tcl Syntax and Specification of Array Bits and Slices	16
SystemVerilog Scope Resolution Operator	17
Specifying Names.....	18
Environment Variables and Pathnames	20
Name Case Sensitivity	20
Extended Identifiers	20
Wildcard Characters	22
Supported Commands.....	22
Using the WildcardFilter Preference Variable	23
Simulator Variables	26
Simulation Time Units.....	27
Optionsets	27
Argument Files	28
Command Shortcuts.....	29
Command History Shortcuts	30
Numbering Conventions	31
VHDL Numbering Conventions.....	31
Verilog Numbering Conventions	32
GUI_expression_format.....	33
Expression Typing	33
Expression Syntax.....	34
Signal and Subelement Naming Conventions	40
Grouping and Precedence	40
Concatenation of Signals or Subelements	40
Record Field Members	42
Searching for Binary Signal Values in the GUI	43
Chapter 2	
Commands.....	45
abort	65
add dataflow.....	66
add list	68
add memory.....	72
add message.....	74
add watch.....	76
add wave	77

add_cmdhelp	84
alias	86
archive load	87
archive write	88
batch_mode	89
bd	90
bookmark add wave	91
bookmark delete wave	93
bookmark goto wave	94
bookmark list wave	95
bp	96
call	102
cd	106
change	107
classinfo ancestry	109
classinfo descriptive	110
classinfo find	112
classinfo implements	114
classinfo instances	116
classinfo interfaces	119
classinfo isa	121
classinfo report	122
classinfo stats	124
classinfo trace	126
classinfo types	128
configure	130
dataset alias	136
dataset clear	137
dataset close	139
dataset config	140
dataset current	142
dataset info	143
dataset list	144
dataset open	145
dataset rename	146
dataset restart	147
dataset save	148
dataset snapshot	149
delete	152
describe	153
disablebp	154
do	155
drivers	157
dumplog64	159
echo	160
edit	161
enablebp	162
encoding	163
environment	164

Table of Contents

examine	166
exit	173
find	174
find connections	180
find infiles	181
find insource	182
force	184
formatTime	190
gc configure	191
gc run	193
help	194
history	195
layout	196
log	198
lshift	201
l sublist	202
mem compare	203
mem display	204
mem list	207
mem load	208
mem save	212
mem search	215
modelsim	218
noforce	219
nolog	220
notepad	222
noview	223
nowhen	224
onbreak	225
onElabError	227
onerror	228
onfinish	230
pause	231
precision	232
printenv	233
process report	234
project	235
pwd	238
quietly	239
quit	240
radix	241
radix define	244
radix delete	248
radix list	249
radix names	250
radix signal	251
readers	252
report	253
restart	255

resume	257
run	258
runStatus	261
searchlog	263
see	266
setenv	267
shift	268
show	269
simstats	270
simstatslist	272
stack down	274
stack frame	275
stack level	276
stack tb	277
stack up	278
status	279
step	280
stop	282
suppress	283
tb	285
Time	286
transcript	290
transcript file	291
transcript path	293
transcript sizelimit	294
transcript wrapcolumn	295
transcript wrapmode	296
transcript wrapwscolumn	297
tssi2mti	298
ui_VVMode	299
unsetenv	301
vcd add	302
vcd checkpoint	305
vcd comment	306
vcd dumports	307
vcd dumportsall	310
vcd dumportsflush	311
vcd dumportslimit	312
vcd dumpportsoff	314
vcd dumpportson	315
vcd file	316
vcd files	318
vcd flush	321
vcd limit	323
vcd off	325
vcd on	326
vcd2wlf	328
vcom	330
vdel	348

Table of Contents

vdir	350
vencrypt	353
verror	359
vgencomp	361
vhencrypt	363
view	367
virtual count	370
virtual define	371
virtual delete	372
virtual describe	373
virtual expand	374
virtual function	375
virtual hide	378
virtual log	379
virtual nohide	381
virtual nolog	382
virtual region	384
virtual save	385
virtual show	386
virtual signal	387
virtual type	391
vlib	393
vlog	396
vmake	423
vmap	425
vsim	427
vsim<info>	468
vsim_break	469
vsource	470
wave	471
wave create	477
wave edit	483
wave export	486
wave import	488
wave modify	489
wave sort	494
when	495
where	503
wlf2log	504
wlf2vcd	506
wlfman	507
wlfrecover	514
write format	515
write list	518
write preferences	519
write report	520
write timing	523
write transcript	525
write tssi	526

write wave 528

Index

End-User License Agreement with EDA Software Supplemental Terms

List of Figures

Figure 2-1. drivers Command Results in Transcript	157
Figure 2-2. find infiles Example	181
Figure 2-3. find insource Example	183
Figure 2-4. readers Command Results in Transcript.....	252

List of Tables

Table 1-1. Conventions for Command Syntax	13
Table 1-2. Examples of Object Names	19
Table 1-3. Wildcard Characters in HDL Object Names	23
Table 1-4. WildcardFilter Arguments	24
Table 1-5. WildcardFilter Argument Groups	25
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 Expressions	34
Table 1-11. Array Constants Supported for GUI Expressions	35
Table 1-12. Variables Supported for GUI Expressions	35
Table 1-13. Array Variables Supported for GUI Expressions	36
Table 1-14. Operators Supported for GUI Expressions	36
Table 1-15. Precedence of GUI Expression Operators	38
Table 1-16. Casting Conversions Supported for GUI Expressions	38
Table 1-17. VHDL Logic Values Used in GUI Search	43
Table 1-18. Verilog Logic Values Used in GUI Search	43
Table 2-1. Supported Commands	45
Table 2-2. Message Viewer Categories	74
Table 2-3. Radix flag Arguments to the Examine Command	169
Table 2-4. runStatus Command States	261
Table 2-5. runStatus -full Command Information	261
Table 2-6. Warning Message Categories for vcom -nowarn	341
Table 2-7. Design Unit Properties	351
Table 2-8. Warning Message Categories for vlog -nowarn	409
Table 2-9. Wave Window Commands for Cursor	472
Table 2-10. Wave Window Commands for Expanded Time Display	473
Table 2-11. Wave Window Commands for Controlling Display	474
Table 2-12. Wave Window Commands for Zooming	475
Table 2-13. when_condition_expression Operators	498

Chapter 1

Syntax and Conventions

This chapter describes the typographical conventions used in this manual to define ModelSim command syntax.

Documentation Conventions	13
File and Directory Pathnames	14
Design Object Names	15
Wildcard Characters	22
Simulator Variables	26
Simulation Time Units	27
Optionsets	27
Argument Files	28
Command Shortcuts	29
Command History Shortcuts	30
Numbering Conventions	31
GUI_expression_format	33

Documentation Conventions

The following conventions are used 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

Table 1-1. Conventions for Command Syntax (cont.)

Syntax notation	Description
...	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 (#), which you can use to add 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

 Command examples do not show either the prompt at the beginning of a line nor the <Enter> keystroke at the end of a line.

File and Directory Pathnames

Several ModelSim commands have arguments that specify file or directory locations (pathnames). For example, the -y argument to the vlog command 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 quotation marks. 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, where various 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

Object Name Syntax	15
Tcl Syntax and Specification of Array Bits and Slices	16
SystemVerilog Scope Resolution Operator	17
Specifying Names	18
Environment Variables and Pathnames	20
Name Case Sensitivity	20
Extended Identifiers	20

Object Name Syntax

To specify object names in ModelSim, you use the following syntax:

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

where

- **dataset_name** — 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 “[Recording Simulation Results With Datasets](#)” in the User’s Manual for more information.
- **datasetSeparator** — The character used to terminate the dataset name. The default is colon (:), although you can specify a different character (except for a backslash (\)) as the dataset separator by using the DatasetSeparator variable in the *modelsim.ini* file. (Refer to [DatasetSeparator](#) in the User’s Manual.) This character must be different than the pathSeparator character.
- **pathSeparator** — The character used to separate hierarchical object names. The default path separator for both VHDL and Verilog is a forward slash (/), although you can specify a different character (except for a backslash (\)) by using the PathSeparator variable in the *modelsim.ini* file. (Refer to [PathSeparator](#) in the User’s Manual.) This character must be different than the datasetSeparator.

Neither (.) nor (/) can be used when referring to the contents of a SystemVerilog package or class.

- **hierarchicalPath** — 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/clk*.
- **objectName** — The name of an object in a design.
- **elementSelection** — Some combination of the following:
 - **Array indexing**— Single array elements are specified using either parentheses (()) or square brackets ([]) around a single number. You must also surround the object and specified array element with curly braces ({}). Refer to [Tcl Syntax and Specification of Array Bits and Slices](#) for important information about using square brackets and parentheses in ModelSim commands.
 - **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 (:). You must also surround the object and specified array slice with curly braces ({}). Refer to [Tcl Syntax and Specification of Array Bits and Slices](#) for important information about using square brackets and parentheses in ModelSim commands.
 - **Record field selection**— A record field is specified using a period (.) followed by the name of the field.

Tcl Syntax and Specification of Array Bits and Slices

Because ModelSim is based on the Tcl scripting language, you must surround objects and signals with braces ({}) when specifying array bits or slices with parentheses (()), spaces, or square brackets ([]).

For example:

```
toggle add {data[3:0]}
toggle add {data(3 to 0)}
force {bus1[1]} 1
```

Further Details

Because ModelSim is based on Tcl, its commands follow Tcl syntax. One problem you may encounter with ModelSim commands is the use of square brackets ([]), parentheses (()), or spaces when specifying array bits and slices. As noted, square brackets specify slices of arrays (for example, `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 do not want this type of behavior when specifying an array slice, so you would use brace escape characters, as follows:

```
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)}
```

Refer to [Tcl Command Syntax](#) in the User's Manual for more information.

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 1-1. 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 examples of the [examine](#) command 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

ModelSim distinguishes 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 (such as 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 (such as *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 (for example, */top/u1/clk*). There is a special case of a fully-rooted name where the top-level design unit name can be unspecified (such as */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 (such as *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.

The current context is also the activation level of an automatic task, function, or block. Different levels of activation may be selected by using the Call Stack window, or by using the 'stack up' or 'stack down' commands.

For example, when you set a breakpoint on line 5 of the following code:

```
package p;
    int I;
    function automatic int factorial(int n);
        if(n==0)
            return 1;
        else
            return n * factorial(n - 1);
    endfunction : factorial
endpackage : p

module top;
    initial begin
        p::I=p::factorial(3);

        $display(p::I);
        $display(p::factorial(4));
    end
endmodule: top
```

When you issue the command:

examine n

the transcript returns:

0

However, when you issue the command:

stack up;examine n

the transcript returns:

1

Table 1-2 contains examples of various ways of specifying object names.

Table 1-2. Examples of Object Names

Object Name	Description
clk	specifies the object clk in the current context
/top/clk	specifies the object clk in the top-level design unit.
/top/block1/u2/clk	specifies the object clk , two levels down from the top-level design unit
block1/u2/clk	specifies the object clk, two levels down from the current context

Table 1-2. Examples of Object Names (cont.)

Object Name	Description
array_sig[4]	specifies an index of an array object
{array_sig(1 to 10)}	specifies a slice of an array object in VHDL; see Tcl Syntax and Specification of Array Bits and Slices for more information
{mysignal[31:0]}	specifies a slice of an array object in Verilog; see Tcl Syntax and Specification of Array Bits and Slices for more information
record_sig.field	specifies a field of a record
“/u_top/ u_sym_wrapper/\br/>sym_row[0]”	specifies an object where the Verilog hierarchy contains escape characters by using quotations.

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](#) in the User’s Manual 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.

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 in many simulator commands.

Supported Commands	22
Using the WildcardFilter Preference Variable	23

Supported Commands

There are a number of commands that support wildcard characters.

The following is a partial list of the commands:

- add dataflow
- add list
- add memory
- add watch
- add wave
- describe
- dumports
- examine
- find (see the Examples section in the [find](#) command for wildcard searches in foreach loops to be applied with commands that do not accept wildcards.)
- log
- vcd add

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
- SystemVerilog multi-dimensional arrays and class objects
- Signals in cells
- Non-dynamic objects of a size equal to or greater than the level specified in the [WildcardSizeThreshold](#) *modelsim.ini* variable, if the variable has been enabled. Refer to [WildcardSizeThreshold](#) and [WildcardSizeThresholdVerbose](#) in the User's Manual for more information.

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 these 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 only with the find command

Note

 A wildcard character does not match a path separator. For example, /dut/* will match /dut/siga 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. The WildcardFilter preference variable is a Tcl List and can be modified using Tcl commands.

The default object types are defined with the WildcardFilter *modelsim.ini* variable and load at each invocation of the simulator. (Refer to [WildcardFilter](#) in the User’s Manual for more information.) You can add both individual ([Table 1-4](#)) and group objects ([Table 1-5](#)) to the current variable list, and you can remove individual objects from the current list.

Determining the Current WildcardFilter Variable Settings

Enter one of the following commands:

set WildcardFilter

or

echo \$WildcardFilter

which returns the list of currently set variables.

Changing the WildcardFilter Settings from the Command Line

Refer to the list of WildcardFilter arguments in [Table 1-4](#) and [Table 1-5](#) to determine what you want to include in the wildcard matches.

- To define a new list of values enter the following command:

```
set WildcardFilter "<arg1 arg2 ...>"
```

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

- To add one or more values to the current list enter the following command:

```
lappend WildcardFilter <arg1 arg2 ...>
```

Note that you must not enclose the space-separated list of arguments in quotation marks.

- To remove a value from the filter use the set command with the Tcl lsearch command to create the new list from the existing list. For example:

```
set WildcardFilter [lsearch -not -all -inline $WildcardFilter Endpoint]
```

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.

Refer to the Tcl man pages (**Help > Tcl Man Pages**) for more information about the lsearch and set commands.

Changing the default WildcardFilter settings

1. Open the modelsim.ini file for editing (refer to [Making Changes to the modelsim.ini File](#) in the User's Manual).
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. Edit the WildcardFilter variable
4. Save the modelsim.ini file to your working directory.

WildcardFilter Argument Descriptions

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

Table 1-4. WildcardFilter Arguments

Argument	Description
Alias	VHDL Alias

Table 1-4. WildcardFilter Arguments (cont.)

Argument	Description
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
Compare	Waveform comparison signal
Constant	VHDL constant
Generic	VHDL generic
ImmediateAssert	VHDL immediate assertions
Integer	VHDL integer
Memory	Verilog memories
NamedEvent	Verilog named event
Net	Verilog net
Parameter	Verilog parameter
Real	Verilog real registers
Reg	Verilog register
Signal	VHDL signal
SpecParam	Verilog specparam
Time	Verilog time registers
Transaction	Transaction stream and stream arrays
Variable	VHDL shared variables in packages and design units.
VHDLFile	VHDL files
VirtualExpr	Virtual expression
VirtualSignal	Virtual signal

Table 1-5 provides a list of the group aliases of WildcardFilter arguments. You can set a group value with the set command. The expanded list of values is returned.

Table 1-5. WildcardFilter Argument Groups

Group Argument	Specific arguments included
AllVHDL	Architecture, Block, Generate, Package, Foreign, Process, Signal, Variable, Constant, Generic, Alias, Subprogram, VHDLFile

Table 1-5. WildcardFilter Argument Groups (cont.)

Group Argument	Specific arguments included
AllVerilogVars	Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, ClassReference
AllVerilog	Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, Class, Cross, Covergroup, Coverpoint, ClassReference
VirtualSignals	VirtualSignal, VirtualExpr
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
AllSignals	Signal, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, VirtualSignal, VirtualExpr, Endpoint, ClassReference
AllSignalsVars	Signal, Variable, Constant, Generic, Alias, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, VirtualSignal, VirtualExpr, Endpoint, ScVariable, ClassReference
AllConstants	Constant, Generic, Parameter, SpecParam
Default	Variable, Constant, Generic, Parameter, SpecParam, Memory, Assertion, Cover, Endpoint, ScVariable, CellInternal, ImmediateAssert VHDLFile

Simulator Variables

You can reference ModelSim variables in a simulator command 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.

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. Refer to [UserTimeUnit](#) in the User's Manual for more information.

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.

Optionsets

By defining and calling optionsets, you can easily use and combine common command line options.

The executable expands these optionsets and passes them to the tool as if they appeared directly on the command line. The behavior is similar to the `-f <file>` option.

Defining an Optionset

Define your optionsets in the [DefineOptionset] section of the `modelsim.ini` file, where the syntax is:

```
<optionset_name> = <command_arguments>
• <optionset_name> — a string that begins with a letter, and contains only letters, numbers, or underscores. The name is case-insensitive.
• <command_arguments> — a list of arguments as you would specify them on the command line. This list of arguments can:
    ○ Refer to another <optionset_name>, enclosed in percent-signs (%).
    ○ Include shell environment variables, preceded by a dollar-sign ($). If you embed the variable in a string, you must surround it with parentheses.
```

You can instruct the executable to return all the values of any optionsets as they are read with the following entry in the [optionsets] section.

```
PRINT_OPTIONSET_VALUE = 1
```

Calling an Optionset

Call your defined optionsets with the -optionset argument to the commands: vlog, vcom and vsim.

The syntax of -optionset is:

```
<command> -optionset <optionset_name>
```

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:

- vlog
- vcom
- vencrypt
- vmake
- vsim

The -f <filename> argument specifies a file that contains additional command line arguments. The following conventions describe 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
```

- Quotation marks (" ") — 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:
 - 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 a backslash (\), which is the line continuation character. You must use a space before the backslash.
 - 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.
 - All text in a line beginning with # to its end is treated as a comment.

Command Shortcuts

The following shortcut techniques are available on the command line.

- You can abbreviate command syntax, but the minimum number of characters required to run a command are those that make it unique. This means the addition of new commands may prevent an older shortcut from working. 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.
- You can enter multiple commands 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
```

Although it seems as if the simstats results should display in the Transcript window, they do 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 (for example, for this prompt: VSIM 12>, n =12)
!<string>	shows a list of executed commands that start with <string>; Use the up and down arrows to choose from the list
!abc	repeats the most recent command starting with "abc"
^xyz^ab^	replaces "xyz" in the last command with "ab"
up arrow and down arrow keys	scrolls through the command history
Ctrl-N (UNIX only)	scroll to the next command
Ctrl-P (UNIX only)	scroll to the previous command
click prompt	left-click 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	31
Verilog Numbering Conventions	32

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 hyphen (-) can also 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 as a negative sign, be sure to enclose the number in quotation marks. For instance, you would type "-0110--" as opposed to -0110--. If you do not include the quotation marks, 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

Table 1-8. VHDL Number Conventions: Style 2 (cont.)

Element	Description
"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 hyphen (-) can also 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 do not include the double quotes, ModelSim will read the '-' as a negative sign. For example:

<pre>'b11111110 'Hffca23 -23749</pre>	<pre>8'b11111110 21'H1fca23</pre>
---------------------------------------	-----------------------------------

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:

configure	examine
	searchlog
	virtual function
	virtual signal
.	
Expression Typing	33
Expression Syntax	34
Signal and Subelement Naming Conventions	40
Grouping and Precedence.....	40
Concatenation of Signals or Subelements	40
Record Field Members	42
Searching for Binary Signal Values in the GUI	43

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.

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)
```

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

Table 1-10. Constants Supported for GUI Expressions (cont.)

Type	Values
single bit constants	expressed as any of the following: 0 1 x X 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 '-') Example: 12'hc91 (This is the preferred notation because it removes the ambiguity about the number of bits.)
Based notation	0x..., 0X..., 0o..., 0O..., 0b..., OB... 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: <ul style="list-style-type: none"> • 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
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	<ul style="list-style-type: none"> • VHDL signals of type bit_vector or std_logic_vector • Verilog register • Verilog net array <p>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]</p>

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 (for example, #-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	Kind
+	arithmetic add	arithmetic
/	arithmetic divide	arithmetic
mod/MOD	arithmetic modulus	arithmetic
*	arithmetic multiply	arithmetic
rem/REM	arithmetic remainder	arithmetic
-	arithmetic subtract	arithmetic
&	concat	arithmetic
<name>'delayed(<time>)	delayed signal (<time>)	attributes
<name>'falling	Falling edge	attributes
<name>'rising	Rising edge	attributes

Table 1-14. Operators Supported for GUI Expressions (cont.)

Operator	Description	Kind
<name>'event	Value change	attributes
<name>'hasX	Value has an X	attributes
and, AND	bitwise and	bitwise logical
nand, NAND	bitwise nand	bitwise logical
nor, NOR	bitwise nor	bitwise logical
or, OR	bitwise or	bitwise logical
xnor, XNOR	bitwise xnor	bitwise logical
xor, XOR	bitwise xor	bitwise logical
rol, ROL	rotate left	bitwise logical
ror, ROR	rotate right	bitwise logical
sla, SLA	shift left arithmetic	bitwise logical
sll, SLL	shift left logical	bitwise logical
sra, SRA	shift right arithmetic	bitwise logical
srl, SRL	shift right logical	bitwise logical
not, NOT, ~	unary bitwise inversion	bitwise logical
&&	boolean and	boolean
!	boolean not	boolean
	boolean or	boolean
==	equal	boolean
====	exact equal ¹	boolean
!==	exact not equal	boolean
>	greater than	boolean
>=	greater than or equal	boolean
<	less than	boolean
<=	less than or equal	boolean
!=, /=	not equal	boolean
&<vector_expr>	AND reduction	reduction
<vector_expr>	OR reduction	reduction
^<vector_expr>	XOR reduction	reduction

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

Table 1-15. Precedence of GUI Expression Operators

Operator	Kind
delayed(), 'falling, 'rising, 'event, 'hasX	attributes
&, , ^	unary
!, not, NOT, ~	boolean
/, mod, MOD, *, rem, REM	arithmetic
nand, NAND, nor, NOR	bitwise logical
and, AND	bitwise logical
xor, XOR, xnor, XNOR	bitwise logical
or, OR	bitwise logical
+, -	arithmetic
&	concat
rol, ROL, ror, ROR, sla, SLA, sll, SLL, sra, SRA, srl, SRL	bitwise logical
>, >=, <, <=	boolean
==, ===, !=, /=	boolean
&&	boolean
	boolean

Note



Arithmetic operators use the std_logic_arith package.

Casting

Table 1-16. 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

Table 1-16. Casting Conversions Supported for GUI Expressions (cont.)

Casting	Description
(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.

Examples in Verilog and VHDL syntax:

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

Grouping and Precedence

Operator precedence generally follows that of the C language, but liberal use of parentheses is recommended.

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, leaving full arrays as elements in the result is not supported. (Please contact Mentor Graphics 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, ModelSim will create the concatenation with a descending index range from (n-1) down to 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_descending) <concatenationExpr>
```

The concat_descending 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_descending) <concatenationExpr>
```

The concat_sort_wild_descending 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_descending) &{ "mybusbasename*" }
```

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

```
(concat_descending) ((concat_sort_wild_descending) &{ "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_ulegic 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 1-17. 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 1-18. 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

Chapter 2

Commands

This chapter describes ModelSim commands that you can enter either on the command line of the Main window or in a DO 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 and whether the command is supported for use in batch simulation mode (`vsim -batch`), and/or command-line mode (`vsim -c`). Refer to [General Modes of Operation](#) in the User's Manual for more information about batch and command-line simulation. 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	-batch	-c
abort	This command halts the execution of a DO file interrupted by a breakpoint or error.	Y	Y
add dataflow	This command adds the specified process, signal, net, or register to the Dataflow window. Wildcards are allowed.	N	N
add list	This command adds the following objects and their values to the List window:	Y	Y
add log	also known as the log command; see “ log ” on page 198	Y	Y
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.	N	N
add message	This command is used within a DO file or script and specifies a user defined runtime message that is sent to the transcript and .wlffiles. Messages are displayed in the Message Viewer window in the GUI. Refer to “Message Viewer Window for information.	Y	Y
add watch	This command adds signals and variables to the Watch window in the Main window.	N	N

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
add wave	This command adds the following objects to the Wave window:	Y	Y
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.	N	Y
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.	Y	N
archive load	The archive load command allows you to load an archived debug database (.dbar) file that was previously created with the archive write command. The archived file may include a number of WLF files, design source files, and a DBG file.	N	N
archive write	The archive write command allows you to create a debug archive file, with the file extension .dbar, that contains one or more WLF files, debug information captured from the design library, an optional connectivity debug database file, and optional HDL source files. With this archived file, you can perform post-simulation debugging in different location from that which the original simulation was run.	N	N
batch_mode	This command returns “1” if Questa SIM is operating in batch mode, otherwise it returns “0.” It is typically used as a condition in an if statement.	Y	Y
bd	This command deletes a breakpoint. You can delete multiple breakpoints by specifying separate information groupings on the same command line.	Y	Y
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.	N	N
bookmark delete wave	This command deletes bookmarks from the specified Wave window.	N	N
bookmark goto wave	This command zooms and scrolls a Wave window using the specified bookmark.	N	N

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
bookmark list wave	This command displays a list of available bookmarks in the Transcript window.	N	N
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.	Y	Y
call	This command calls the following types of functions/tasks.	Y	Y
change	This command modifies the value of a: VHDL constant, generic, or variable; Verilog register or variable; or C variable if running C Debug.	Y	Y
classinfo ancestry	This command returns class inheritance hierarchy for a named class type.	Y	Y
classinfo descriptive	This command returns the descriptive class name for the specified authoritative class name.	Y	Y
classinfo find	This command reports on the current state of a specified class instance, whether it exists, has not yet been created, or has been destroyed.	Y	Y
classinfo implements	This command displays a list of which classes implement SystemVerilog interface classes. The type of the class argument affect the contents of this list.	Y	Y
classinfo instances	This command reports the list of existing class instances of a specific class type. You can use this to determine what class instances to log or examine. It can also help in debugging problems where class instances are not being cleaned up as they should be resulting in excessive memory usage.	Y	Y
classinfo interfaces	This command lists the interface class types that match or do not match a specified pattern. Finds all interface classes that match a regular expression and determines the full path of interface class types.	Y	Y
classinfo isa	This command returns to the transcript a list of all classes extended from the specified class type.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
classinfo report	This command prints detailed reports on class instance usage. The command displays columns for class type names and their current, peak and total class instance counts. The columns may be arranged, sorted, or eliminated using the command arguments.	Y	Y
classinfo stats	This command prints statistics about the total number of class types and total, peak, and current class instance counts during the simulation.	Y	Y
classinfo trace	This command displays the active references to the specified class instance. This is very useful in debugging situations where class instances are not being destroyed as expected because something in the design is still referencing them. Finding those references may lead to uncovering bugs in managing these class references which often lead to large memory savings.	Y	Y
classinfo types	This command lists the class types that match or do not match a specified pattern. Finds all classes that match a regular expression and determines the full path of class types.	Y	Y
configure	The configure command invokes the List or Wave widget configure command for the current default List or Wave window.	N	Y
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.	N	Y
dataset clear	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.	N	Y
dataset close	This command closes an active dataset. To open a dataset, use the dataset open command.	N	Y
dataset config	This command configures WLF parameters for an open dataset and all aliases mapped to that dataset.	N	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
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.	N	Y
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.	N	Y
dataset list	This command lists all active datasets.	N	Y
dataset open	This command opens a WLF file (either the currently running vsim.wlf or a saved WLF file) and/or UCDB file (representing coverage data) and assigns it the logical name that you specify.	N	Y
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.	N	Y
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.	N	Y
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.	N	Y
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.	N	Y
delete	This command removes objects from either the List or Wave window. Arguments to this command are order dependent.	N	Y
describe	This command displays information about simulation objects and design regions in the Transcript window.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
disablebp	This command turns off breakpoints and when commands. To turn on breakpoints or when commands again, use the enablebp command.	Y	Y
do	This command executes the commands contained in a DO file.	Y	Y
drivers	This command displays the names and strength of all drivers of the specified object.	Y	Y
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.		
echo	This command displays a specified message in the Transcript window.	Y	Y
edit	This command invokes the editor specified by the EDITOR environment variable. By default, the specified filename will open in the Source window.	N	N
enablebp	This command turns on breakpoints and when commands that were previously disabled.	Y	Y
encoding	This command translates between the 16-bit Unicode characters used in Tcl strings and a named encoding, such as Shift-JIS.	Y	Y
environment	This command has two forms, environment and env. It allows you to display or change the current dataset and region/signal environment.	Y	Y
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.	Y	Y
exit	This command exits the simulator and the ModelSim application.	Y	Y
find	This command locates objects by type and name. Arguments to the command are grouped by object type.	N	Y
find connections	This command returns the set of nets that are electrically equivalent to a specified net. It is only available during a live simulation.	N	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
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.	Y	Y
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.	Y	Y
force	This command allows you to apply stimulus interactively to VHDL signals, Verilog nets and registers.	Y	Y
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.	Y	Y
gc configure	This command specifies when the System Verilog garbage collector will run. The garbage collector may be configured to run after a memory threshold has been reached, after each simulation run command completes, and/or after each simulation step command. The default settings are optimized to balance performance and memory usage for either regular simulation or class debugging (vsim -classdebug). Returns the current settings when specified without arguments.	N	Y
gc run	This command runs the SystemVerilog garbage collector.	N	Y
help	This command displays in the Transcript window a brief description and syntax for the specified command.	N	Y
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).	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
layout	This command allows you to perform a number of editing operations on custom GUI layouts, such as loading, saving, maximizing, and deleting.	N	N
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 vsim -wlf option of the vsim command or by setting the WLFFilename variable in the modelsim.ini file.	Y	Y
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.	Y	Y
l sublist	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.	Y	Y
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.	Y	Y
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.	Y	Y
mem list	This command displays a flattened list of all memory instances in the current or specified context after a design has been elaborated.	Y	Y
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.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
<code>mem save</code>	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.	Y	Y
<code>mem search</code>	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.	Y	Y
<code>modelsim</code>	The modelsim command starts the ModelSim GUI without prompting you to load a design.	?	?
<code>noforce</code>	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.	Y	Y
<code>nolog</code>	This command suspends writing of data to the wave log format (WLF) file for the specified signals.	Y	Y
<code>notepad</code>	This command opens a simple text editor. It may be used to view and edit ASCII files or create new files.	N	N
<code>noview</code>	This command closes a window in the ModelSim GUI. To open a window, use the view command.	N	N
<code>nowhen</code>	This command deactivates selected when commands.	Y	Y
<code>onbreak</code>	This command is used within a DO file and specifies one or more scripts to be executed when running a script that encounters a breakpoint in the source code.	Y	Y
<code>onElabError</code>	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 DO file script. Use the onElabError command without arguments to return to a prompt.	Y	Y
<code>onerror</code>	This command is used within a DO file script before a run command; it specifies one or more commands to be executed when a running script encounters an error.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
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.	Y	Y
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.	Y	Y
precision	This command determines how real numbers display in the graphic interface (for example, 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.	Y	Y
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.	Y	Y
process report	This command creates a textual report of all processes displayed in the Process Window.	Y	Y
project	This command is used to perform common operations on projects.	N	Y
pwd	This Tcl command displays the current directory path in the Transcript window.	Y	Y
quietly	This command turns off transcript echoing for the specified command.	Y	Y
quit	This command exits the simulator.	Y	Y
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.	Y	Y
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 or setup a fixed or floating point radix. User-defined radices are available for use in the most windows and with the examine command.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
radix delete	This command will remove the radix definition from the named radix.	Y	Y
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.	Y	Y
radix names	This command returns a list of currently defined radix names.	Y	Y
radix signal	This command sets or inspects radix values for the specified signal in the Objects, Locals, Schematic, and Wave windows. When no argument is used, the radix signal command returns a list of all signals with a radix.	Y	Y
report	This command displays information relevant to the current simulation.	Y	Y
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.)	Y	Y
resume	This command is used to resume execution of a macro (DO) file after a pause command or a breakpoint.	Y	Y
run	This command advances the simulation by the specified number of timesteps.	Y	Y
runStatus	This command returns the current state of your simulation to stdout after issuing a run or step command.	Y	Y
searchlog	This command searches one or more of the currently open logfiles for a specified condition.	N	Y
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.	Y	Y
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.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
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.	Y	Y
show	This command lists HDL objects and subregions visible from the current environment.	Y	Y
simstats	This command returns performance-related statistics about elaboration and simulation. The statistics measure the simulation kernel process 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.	Y	Y
simstatslist	This command returns performance-related statistics about elaboration and simulation. Use this command in place of the simstats command to produce the original statistics output format as a list instead of on separate lines.	Y	Y
stack down	This command moves down the call stack.	Y	Y
stack frame	This command selects the specified call frame.	Y	Y
stack level	This command reports the current call frame number.	Y	Y
stack tb	This command 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. The stack tb command is an alias for the tb command.	Y	Y
stack up	This command moves up the call stack.	Y	Y
status	This command lists summary information about currently interrupted macros.	Y	Y
step	The step command is an alias for the run command with the -step switch. Steps the simulator to the next HDL.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
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.	Y	Y
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.	Y	Y
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.	Y	Y
Time	These commands allow you to perform comparisons between, operations on, and conversions of, time values.	Y	Y
transcript	This command controls echoing of commands executed in a macro file. If no option is specified, the current setting is reported.	Y ¹	Y
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.	Y	Y
transcript path	This command returns the full pathname to the current transcript file.	Y	Y
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.	Y	Y
tssi2mti	This command is used to convert a vector file in TSSI Format into a sequence of force and run commands.	N	Y
unsetenv	This command deletes an environment variable. The deletion is not permanent – it is valid only for the current ModelSim session.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
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.	Y	Y
vcd add	This command adds the specified objects to a VCD file.	Y	Y
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	Y	Y
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.	Y	Y
vcd dumports	This command creates a VCD file that includes port driver data.	Y	Y
vcd dumportsall	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: \$dumportsall	Y	Y
vcd dumportsflush	This command flushes the contents of the VCD file buffer to the specified VCD file. Related Verilog task: \$dumportsflush	Y	Y
vcd dumportslimit	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.	Y	Y
vcd dumpportoff	This command turns off VCD dumping and records all dumped port values as x.	Y	Y
vcd dumpporton	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 dumpportoff. Related Verilog task: \$dumpportson	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
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.	Y	Y
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	Y	Y
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	Y	Y
vcd limit	This command specifies the maximum size of a VCD file (by default, limited to available disk space).	Y	Y
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	Y	Y
vcd on	This command turns on VCD dumping to the specified file and records the current values of all VCD variables.	Y	Y
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.	Y	Y
vcom	The vcom command compiles VHDL source code into a specified working library (or to the work library by default).	Y	Y
vdel	This command deletes a design unit from a specified library. This command provides additional information with the -help switch.	Y	Y
vdir	This command lists the contents of a design library and checks the compatibility of a vendor library. If vdir cannot read a vendor-supplied library, the library may not be compatible with ModelSim.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
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.	Y	Y
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 or -h switch.	Y	Y
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.	Y	Y
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.	Y	Y
view	This command opens the specified window. If you specify this command without arguments it returns a list of all open windows in the current layout.	N	N
virtual count	This command reports the number of currently defined virtuals that were not read in using a macro file.	N	Y
virtual define	This 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.	N	Y
virtual delete	This command removes the matching virtuals.	N	Y
virtual describe	This command prints to the transcript a complete description of the data type of one or more virtual signals. Similar to the existing describe command.	N	Y
virtual expand	This 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.	N	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
virtual function	This 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>.	N	Y
virtual hide	This 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.	N	Y
virtual log	This 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.	N	Y
virtual nohide	This command reverses the effect of a virtual hide command, causing the specified real or virtual signals to reappear the Objects window.	N	Y
virtual nolog	This 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.	N	Y
virtual region	This command creates a new user-defined design hierarchy region.	N	Y
virtual save	This command saves the definitions of virtuals to a file named virtual.do in the current directory.	N	Y
virtual show	This command lists the full path names of all explicitly defined virtuals.	N	Y
virtual signal	This 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>.	N	Y
virtual type	This 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.	N	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
vlib	This command creates a design library. You must use vlib rather than operating system commands to create a library directory or index file.	Y	Y
vlog	The vlog command compiles Verilog source code and SystemVerilog extensions into a specified working library (or to the work library by default). Compressed SystemVerilog source files (those compressed with zlib) are accepted.	Y	Y
vmake	The vmake utility allows you to use a 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.	Y	Y
vmap	The vmap command defines a mapping between a logical library name and a directory by modifying the modelsim.ini file.	Y	Y
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)	Y	Y
vsim<info>	The vsim<info> commands return information about the current vsim executable.	Y	Y
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.	Y	Y
vsources	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..	Y	Y
wave	A number of commands are available to manipulate and report on the Wave window.	N	N
wave sort	This command sorts signals in the Wave window by name or full path name.	N	N
when	This command instructs ModelSim to perform actions when the specified conditions are met.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
where	This command displays information about the system environment. It is useful for debugging problems where ModelSim cannot find the required libraries or support files.	Y	Y
wlf2log	This command translates a ModelSim WLF file (vsim.wlf) to a QuickSim II logfile. It 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.	Y	Y
wlf2vcd	This command translates a ModelSim WLF file to a standard VCD file. Complex data types that are unsupported in the VCD standard (records, memories, and so on) are not converted.	Y	Y
wlfman	This command allows you to get information about and manipulate saved WLF files.	Y	Y
wlfrecover	This command attempts to "repair" WLF files that are incomplete due to a crash or if the file was copied prior to completion of the simulation. Use this command if you receive a "bad magic number" error message when opening a WLF file. You can run the command from the VSIM> or ModelSim> prompt or from a shell.	Y	Y
write format	This 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.	N	Y
write list	This command records the contents of the List window in a list output file.	N	Y
write preferences	This 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.	N	Y
write report	This 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.	Y	Y

Table 2-1. Supported Commands (cont.)

Command name	Action	-batch	-c
write timing	This command displays path delays and timing check limits, unadjusted for delay net delays, for the specified instance.	Y	Y
write transcript	This command writes the contents of the Transcript window to the specified file. The resulting file can then be modified to replay the transcribed commands as a DO file (macro).	N	Y
write tssi	This command records the contents of the List window in a "TSSI format" file.	Y	Y
write wave	This command records the contents of the Wave window in PostScript format. The output file can then be printed on a PostScript printer.	N	N

1. transcript on | off only are supported.

abort

Halts the execution of a DO file interrupted by a breakpoint or error.

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

Syntax

abort [<n> | all]

Arguments

- <n>
(optional) The number of nested DO file script 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 DO files.

add dataflow

Adds the specified process, signal, net, or register to the Dataflow window. Wildcards are allowed.

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. Must be specified as the first argument to the add dataflow command. Wildcards are allowed. Multiple objects are specified as a space separated list, Refer to the section “[Wildcard Characters](#)” on page 22 for wildcard usage as it pertains to the add commands.
- -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 \[ModelSim User's Manual\]](#)

[Dataflow Window \[ModelSim GUI Reference Manual\]](#)

add list

Adds objects and their values to the List window.

Syntax

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

Description

Use add list to display the following types of objects and their values in the List window:

- VHDL signals and variables
- Verilog nets and registers
- User-defined buses

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.

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

Arguments

- <object> ...

(required, if <object_name>{sig ...} is not specified.) Specifies the name of the object to be listed. When you use this argument, you must specify it as the first argument to the add list command. Enter multiple objects as a space separated list. Wildcards are allowed. Refer to the section “[Wildcard Characters](#)” for wildcard usage as it pertains to the add commands.

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
```

- <object_name> {sig ...}

(required, if <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. When you use this argument, you must specify it as the first argument to the add list command. You must enclose the full argument set in braces ({}).

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.

- -filter <f> | -nofilter <f>

(optional) Allows a one-time modification of the WildcardFilter in the command invocation. The add list command can take as many [-filter <f>] and [-nofilter <f>] arguments as you want to specify. Valid filters, <f>, are exactly the same set of words that you can apply to the WildcardFilter. The order filters are applied during a command is WildcardFilter first, then any user specified filters. The -filter values are added to the filter, the -nofilter values are removed from the filter. The filters are applied in the order specified so conflicts are resolved with the last specified wins.

- -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.

- **-nodelta**

(optional) Specifies that the delta column not be displayed when adding signals to the List window. Identical to [configure](#) list -delta none.

- **-<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, sfixed, symbolic, ufixed, 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. (Refer to [DefaultRadix](#) in the User's Manual.)

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, sfixed, symbolic, time, ufixed, and default.

Refer to the [radix](#) command for information about sfixed and ufixed radix types.

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

- **-radixenumnumeric**

This option overrides the global setting of the default radix (the variable in the *modelsim.ini* file).

- **-radixenumsymbolic**

(optional) Reverses the action of -radixenumnumeric and sets the global setting of the default radix (the 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 | -notrigger**

(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. The maximum width, when not specifying this argument is 30,000 characters, which you can override with this switch.

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

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

Related Topics

[add wave](#)

add memory

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

Refer to “Wildcard Characters” for wildcard usage as it pertains to the add commands.

Syntax

```
add memory [-addressradix {decimal | hex}] [-dataradix <type>]  
[-radixenumnumeric | -radixenumsymbolic] [-wordsperline <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, sfixed, symbolic, time, ufixed, and default.

Refer to the [radix](#) command for information about sfixed and ufixed radix types.

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. (Refer to [DefaultRadix](#) in the User’s Manual.) 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 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 enums as symbols by reversing the action of the -radixenumnumeric option.
- **-wordsperline <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. Must be specified as the final argument to the add memory command. Multiple memories are specified as a space separated list.

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 \[ModelSim GUI Reference Manual\]](#)

add message

Used within a DO file or script to specify a user-defined runtime message to be sent to the transcript and .wlf files. Messages are displayed in the Message Viewer window in the GUI. Refer to the *GUI Reference Manual* for more information on this window.

Syntax

```
add message <message_body> [-category <category>] [-efftime <time>] [-file <filename>]  
[-id <id_number>] [-inline] [-line <linenumber>] [-noident] [-nolevel] [-objects <list>]  
[-region region] [-severity {error | note | warning}]
```

Arguments

- <message_body>
(required) User specified message.
- -category <category>
(optional) Sets the category for the message in the Message Viewer window, where the default is USER. The Message Viewer window Category column recognizes the following keywords:

Table 2-2. Message Viewer Categories

DISPLAY	FLI	PA
PLI	SDF	TCHK
VCD	VITAL	WLF
MISC	USER	<user-defined>

- -efftime <time>
(optional) Specifies the simulation time when the message is saved to the log file. The Message Viewer window Time column lists the time specified when the message is called. Useful for placing messages at specific times in the simulation.

 <time> — Specified as an integer or decimal number.
- -file <filename>
(optional) Displays a user specified string in the File Info column of the Message Viewer window.
- -id <id_number>
(optional) Assigns an identification number to the message.

 <id_number> — Any non-negative integer from 0 - 9999 where the default is 0. The number is added to the base identification number of 80000.

- **-inline**
(optional) Causes the message to also be returned to the caller as the return value of the add message command.
- **-line <linenumber>**
(optional) Displays the user specified number in File Info column of the Message Viewer window.
- **-noident**
(optional) Prevents return of the ID number of the message.
- **-nolevel**
(optional) Prevents return of the severity level of the message.
- **-objects <list>**
(optional) List of related design items shown in the Objects column of the Message Viewer window.
 <list> — A space separated list enclosed in curly braces ({}) or quotation marks (" ").
- **-region region**
(optional) Displays the message in the Region column of the Message Viewer window.
- **-severity {error | note | warning}**
(optional) Sets the message severity level.
 error — ModelSim cannot complete the operation.
 note — (default) The message is informational.
 warning — There may be a problem that will affect the accuracy of the results.

Examples

- Create a message numbered 80304.

add message -id 304 <message>

Related Topics

- [displaymsgmode \[ModelSim User's Manual\]](#)
[msgmode \[ModelSim User's Manual\]](#)
[Message Viewer Window \[ModelSim GUI Reference Manual\]](#)

add watch

Adds signals and variables to the Watch window in the Main window.

Refer to “Wildcard Characters” for wildcard usage as it pertains to the add commands.

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 the first argument to the add watch command.

Wildcard characters are allowed. (Note that the WildcardFilter Tcl preference variable identifies the types to ignore when matching objects with wildcard patterns.) Wildcard expansion is limited to 150 items. If you exceed this limit, a dialog box will ask you to accept the limit or cancel the operation.

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, sfixed, symbolic, time, ufixed, and default.

Refer to the [radix](#) command for information about sfixed and ufixed radix types.

You can change the default radix for the current simulation with the [radix](#) command. You can change the default radix permanently by editing the DefaultRadix variable in the *modelsim.ini* file. (Refer to [DefaultRadix](#) in the User’s Manual.)

- -radixenumnumeric

(optional) Causes Verilog 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 enums as symbols by reversing the action of the -radixenumnumeric option.

Related Topics

[Watch window \[ModelSim GUI Reference Manual\]](#)

[DefaultRadix \[ModelSim User's Manual\]](#)

add wave

Adds objects to the Wave window.

Syntax

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

Description

Use add wave to display the following types of objects in the Wave window:

- VHDL signals and variables
- Verilog nets and registers
- SystemVerilog class objects
- Dividers and user-defined buses.

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

Refer to “[Wildcard Characters](#)” on page 22 for wildcard usage as it pertains to the add commands.

Arguments to this command are order dependent, as described in the argument descriptions.

Arguments

- **-allowconstants**

(optional) *For use with wildcard searches.* Specifies to add constants matching the wildcard search 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
 -depths 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. Enter multiple objects 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 quotation marks ("") 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.
- **-filter <f> | -nofilter <f>**
(optional) Allows a one-time modification of the WildcardFilter in the command invocation. The add list command can take as many [-filter <f>] and [-nofilter <f>] arguments as you want to specify. Valid filters, <f>, are the exact set of words that apply to the WildcardFilter. The WildcardFilter is used first during a command, followed by the user specified filters, if any. The -filter values are added to the filter, the -nofilter values are removed from the filter. They are applied in the order specified, so conflicts are resolved with the last specified wins.
- **-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 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 quotation marks ("") or braces ({ }) if it contains any spaces.

<sig_name> ... — Specifies the signals to add to the group. Enter multiple signals 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.

- **-max <real_num>**
(optional) Specifies the maximum Y-axis data value to display 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 display 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.

- **-noupdate**
(optional) Prevents the Wave window from updating when a series of add wave commands execute in series.
- **-numdynitem <int>**
(optional) Specifies the number of child elements of a queue or dynamic array to display in the Wave window. For example, if you specify the value 3, then only three elements display in the Wave window.

- <int> — Any non-negative integer from 0 to the number of elements of the specified queue or dynamic array.
- -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.
 - -queueends
 - (optional) Adds a SystemVerilog queue to the Wave window and displays the first and last elements of the queue.
 - <queue> — The relative or full path to a queue.
 - -<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, sfixed, symbolic, time, ufixed, and default.

If you do not specify a radix 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. (Refer to [DefaultRadix](#) variable in the User's Manual.)

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. You can use the -radix <type> switch in place of the -<radix_type> switch. For example, -radix hexadecimal is the same as -hex.
 - <type> — binary, ascii, unsigned, decimal, octal, hex, sfixed, symbolic, time, ufixed, and default.
 - Refer to the [radix](#) command for information about sfixed and ufixed radix types.
 - This option overrides the global setting of the default radix (the `variable` in the *modelsim.ini* file) for the current simulation only.
 - -radixenumnumeric
 - (optional) Causes Verilog enums to display 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 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.
- **-startdynitem <int>**
(optional) Specifies the index of a queue or dynamic array from where the Wave window starts displaying the data. For example, if a queue has 10 elements and **-startdynitem 3** is specified, the display starts from q[3].
<int> — Any non-negative integer where 0 is the default.
- **-time**
(optional) Use time as the radix for Verilog objects that are register-based types (register vectors, time, int, and integer types).
- **<object_name> ...**
(required unless specifying {<object_name> {sig1 sig2 ...}}) Specifies the names of objects to include in the Wave window. Must be specified as the final argument to the add wave command. Wildcard characters are allowed. Enter multiple objects as a space separated list. Note that the WildcardFilter Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.

You can add variables by preceding them with 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 the final argument to the add wave command.
sig — A space-separated list, enclosed in braces ({}), of the signals 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 {}**

Related Topics

[add list](#)[Wave Window \[ModelSim GUI Reference Manual\]](#)

add_cmdhelp

Adds the specified command name, description, and command arguments to the command-line help. You can then access the information using the help command.

Note

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

Syntax

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

Arguments

- {<command_name>}
(required) Specifies the command name to enter as an argument to the help command. Must be enclosed in braces ({ }). The command_name cannot be the same as an 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 does not 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} {} {}

alias

Displays or creates user-defined aliases. Any arguments passed on invocation of the alias are passed through to the specified commands.

Returns nothing. Existing commands (for example, run, env, and so forth) cannot be aliased.

Syntax

alias [[**<name>**](#) [[**"<cmds>"**](#)]]

Arguments

- [**<name>**](#)
(optional) Specifies the new procedure name to use when invoking the commands.
- [**"<cmds>"**](#)
(optional) Specifies the command or commands to evaluate when invoking the alias.
Specify multiple commands 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 named "myquit" that, when executed, invokes write list to write the contents of the List window to the file *mylist.save*, and then invokes quit to exit ModelSim.

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

archive load

Enables you to load an archived debug database (.dbar) file that was previously created with the archive write command. The archived file may include a number of WLF files, design source files, and a DBG file.

Syntax

```
archive load <archive_name> [-dbgDir <directory_name>] -wlf <wlf_file_name>
```

Arguments

- <archive_name>
(required) Specifies the name of the archived file to open for reading. A suggested suffix is .dbar.
- -dbgDir <directory_name>
(optional) Specifies a location to extract files into. Files are extracted on demand when ModelSim needs them. If you do not specify this switch, the command extracts to the current working directory.
- -wlf <wlf_file_name>
(required) Specifies the WLF files to open for analysis.

<wlf_file_name> — can be a single file or a list of files. If you use a list of file names, you must enclose it in curly braces { }. The name of the wlf file must exactly match the name and file path (if provided) specified in the archive write command.

Related Topics

[archive write](#)

archive write

Enables you to create a debug archive file, with the file extension .dbar, that contains one or more WLF files, debug information captured from the design library, an optional connectivity debug database file, and optional HDL source files. You can use this archived file to perform post-simulation debugging in a location that differs from where the original simulation was run.

Syntax

```
archive write <archive_name> -wlf <wlf_file_name> [-include_src] [-dbg <dbg_file_name>]
```

Arguments

- <archive_name>
(required) Specifies the name of the archive file to create. A suggested suffix is .dbar.
- -wlf <wlf_file_name>
(required) Specifies the name of the WLF file to use for post-simulation analysis.
 <wlf_file_name> — can be a single file, or a list of files enclosed in curly braces {} if you want to capture more than one WLF file in the archive.
- -include_src
(optional) Indicates to capture source files in the archive. This is off by default, which means no source is captured in the archive.
- -dbg <dbg_file_name>
(optional) Specifies the name of an existing debug database (.dbg) file to include in the archive.

batch_mode

Returns “1” if ModelSim is operating in batch mode, otherwise it returns “0.” 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 works both in and out of batch mode, use the batch_mode command to determine which command to use. For example:

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

Related Topics

[General Modes of Operation \[ModelSim User's Manual\]](#)

bd

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

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 to delete the breakpoint. The filename must match the one used to set the breakpoint, including whether you used a full pathname or a relative name. Must be the first argument to the bd command, if you do not specify <id_number> or <label>.
- <line_number>
(required) A string that specifies the line number of the breakpoint to delete.
- <id_number> | <label>
(required when not specifying <filename>.) Specifies the identification of breakpoints using markers assigned by the [bp](#) command. Must be the first argument to the bd command, if you do not specify <filename>.
 <id_number> — A string that specifies the identification number of the breakpoint to delete. Use the -id argument to the [bp](#) command to set the identification number.
 <label> — A string that specifies the label of the breakpoint to delete. Use the -label switch to the [bp](#) command to set the label.

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 6 alu.vhd 234

Related Topics

[bp](#)

bookmark add wave

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.

Note

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

Syntax

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

Arguments

- <label>

(required) A string that specifies the name for the bookmark. Must be 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 ({}).

You must also enclose the complete grouping of <range_start> and <range_end> 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 specifying the vertical scroll position of the window. You must specify a zoom range before you specify topindex. The number identifies which object to scroll the window to. For example, specifying 20 will scroll the Wave window down to show the 20th object.

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](#)

bookmark delete wave

This command deletes bookmarks from the specified Wave window.

Note

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

Syntax

bookmark delete wave {<label> | -all}

Arguments

- <label> | -all

(required) Controls which bookmarks to delete. Must be the first argument to the bookmark delete wave command.

<label> — Specifies the name of the bookmark to delete.
-all — Specifies to delete all bookmarks in the window.

Examples

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

bookmark delete wave foo

Related Topics

[bookmark add wave](#)

[bookmark goto wave](#)

[bookmark list wave](#)

bookmark goto wave

Zooms and scrolls a Wave window using the specified bookmark.

Note

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

Syntax

`bookmark goto wave <label>`

Arguments

- `<label>`

(required) Specifies the bookmark to go to. Must be the first argument to the bookmark goto wave command.

Related Topics

[bookmark add wave](#)

[bookmark delete wave](#)

[bookmark list wave](#)

bookmark list wave

Displays a list of available bookmarks in the Transcript window.

Syntax

bookmark list wave

Arguments

none

Related Topics

[bookmark add wave](#)

[bookmark delete wave](#)

[bookmark goto wave](#)

bp

Either sets a file-line breakpoint, or returns information about breakpoint(s). Allows condition expressions to use enum names, as well as literal values.

Syntax

Setting an HDL breakpoint

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

Querying a breakpoint

```
bp [-query <filename> [{<start_line_number> <end_line_number>} | <line_number>]]
```

Reporting all breakpoints

```
bp
```

Note

 Specifying bp with no arguments returns a list of all breakpoints in the design, containing information about each breakpoint. For example, the command “bp” might return:

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

Arguments

- <filename>
(optional) Specifies the name of the source file in which to set the breakpoint. If you do not specify a filename, the command uses the source file of the current context.
- <line_number>
(required to set an HDL breakpoint) Specifies the line number on which to set the breakpoint.
- in
(required for task or function breakpoints) Supports the lookup of Verilog and SystemVerilog task and function names as an alternative to file name and line numbers. Places a breakpoint on the first executable line of the specified task or function. Does not work for VHDL or SystemC.
- -ancestor
(optional) Stops the simulation only when an ancestor parent of the process matches the given process-name.
- -appendinst
(optional) When specifying multiple breakpoints with -inst, appends 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. The bp command enables breakpoints by default.

- <command>...

(optional, must be specified as the final argument) Specifies one or more commands to execute 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 also applies if you use a DO file script within the command string.

If you need to enter many commands after the breakpoint, you could place them in a DO file script.

- -cond "<condition_expression>"

(optional) Specifies one or more conditions 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 braces ({}) or quotation marks (" ") when the expression makes use of spaces. Refer to the note below when setting breakpoints in the GUI.

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 behavior differs from 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 a condition expression by choosing **Tools > Breakpoints** from the main menu and entering the expression in the **Breakpoint Condition** field of the **Modify Breakpoints** dialog box. Do not enclose the condition expression in quotation marks (" ") or braces ({}).

The condition expression can use the following operators:

Operator on	Operator Syntax
equals	<code>==, =</code>
not	<code>!=, /=</code>
equal	
AND	<code>&&, AND</code>
OR	<code> , OR</code>

The operands can be object names, `signame`event`, or constants. Subexpressions in parentheses are permitted. The command is 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 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> [{<start_line_number> <end_line_number>} | <line_number>]

(optional) Returns information about the breakpoint(s) and the list of executable lines in the specified file. When specified without an argument, the command returns the list of all breakpoints and its status. Returns nothing if the line number(s) specified in the argument are not executable.

<filename> — The name of the file you want to query for breakpoint(s).

<start_line_number> <end_line_number> — Specifies a range of line numbers. The command returns a list of any executable lines within the range.

<line_number> — The line number you want to query for a breakpoint.

When <line_number> is an argument, then the command returns six fields of information.

For example, the command:

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

Returns:

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

- {1 | 0} — Indicates whether a breakpoint exists at the location.
 - 0 — Breakpoint does not exist.
 - 1 — Breakpoint exists.
- 1 — always reports a 1.

- <file_name>
- <line_number>
- <id_number>
- {1 | 0} — Indicates whether the breakpoint is enabled.
 - 0 — Breakpoint is not enabled.
 - 1 — Breakpoint is enabled.
- -uvm

Specifies UVM-style instance name(s) for setting source breakpoints on class instances in the UVM hierarchy. Must be followed by the -inst <instance_name> option.

Examples

- List all existing breakpoints in the design, including the source file names, line numbers, breakpoint id numbers, 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

- Set a breakpoint at line 153 of the source file of the current context:

bp 153

- Execute the macro.do DO 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 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}
```

- Set a breakpoint on the first executable line of the function /uvm_pkg::set_config_int.

```
bp in /uvm_pkg::set_config_int
```

- List the line number and enabled/disabled status (1 = enabled, 0 = disabled) of all breakpoints in testadd.vhd.

```
bp -query testadd.vhd
```

- List all executable lines in testadd.vhd between lines 2 and 59.

```
bp -query testadd.vhd 2 59
```

- List details about the breakpoint on line 48.

```
bp -query testadd.vhd 48
```

Related Topics

[bd](#)

[Editing File-Line Breakpoints \[ModelSim GUI Reference Manual\]](#)

call

Calls SystemVerilog static functions and class functions (but not tasks) from the vsim command line in live simulation mode. Calls PLI and VPI system tasks and system functions. Returns function return values to the vsim shell as a Tcl string. Returns class references as class instance IDs.

Syntax

Note

 The grouping of arguments can vary depending on how you want the command to perform a search on a system task or function name. For example:

- <pathToFunction> <classInstancePath> — SystemVerilog static functions and class functions
 - -usertf -builtin <systfName> — PLI and VPI system tasks and system functions (restricts the search for the task/function name to either the user-added PLI/VPI routines or to the built-in routines):
-

```
call [-env <hierEnvPath>] [{<pathToFunction> [<classInstancePath>]}] [{-usertf | -builtin <systfName>}] [<arg1> [<arg2>] ...[<argN>]]
```

Arguments

- -env <hierEnvPath>
(optional) Hierarchical environment path to use as the starting scope for the object name lookups. If present, must appear before actual function name.
- <pathToFunction>
(required when calling a System Verilog static or class function) The name of a function, which you can specify by:
 - The path to the function declaration, through the structural hierarchy, or declaration hierarchy. You must specify hierarchical paths must as a full path to a function, or a function that exists relative to the current context (as shown in the Structure window, or returned by the environment command).
 - A class instance hierarchical path.
 - A class instance id string.
- <classInstancePath>
(optional) Must be specified if the function path is a declaration path and the function is a non-static class function. Do not specify the class instance path name if the given function path is a class instance variable reference or a class instance name in the format @<class_type>@nnn, because the class instance information can be extracted from the pathname itself.

- **-builtin**
(optional) Search only built-in system functions or task names. (The \$ is understood to be a prefix.)
- **-userf**
(optional) Search only user-defined system functions or task names. (The \$ is understood to be a prefix.)
- **<systfName>**
(required when calling a system task or function) The system task or function to execute. You can specify the sysfName according to the following syntax rules:
 - `\$<systfName>` (for example, `\$display` or `\$mytask`)
 - With `-userf` or `-builtin` flags (as the case may be). (for example, `-userf mytask` or `-builtin display`)
 - **<systfName>** (for example, `display` or `mytask`) In this case SystemVerilog static functions and class functions are searched first for a match, and then the PLI/VPI system tasks/functions.
- **<arg1> [<arg2>] ...[<argN>]**
(optional) All arguments required by the function are specified in a space-separated list, in declaration order. If a function has default arguments, the arguments can be omitted from the command line, provided that the arguments occur at the end of the declaration list. Function input arguments can be constant values, including integers, enumerated values, and strings. Strings containing spaces or special characters must be enclosed in quotation marks (" ") or braces ({ }), or Tcl will try to interpret the string. For example: "my string" or {my string}. Arguments can also be design objects. Class references can be arguments, specified by either their design instance path or class instance id string. If a function has type inout, out, or ref arguments, suitable user design objects must be passed in as arguments. Any passed in argument will first be tested to determine if it is an appropriate constant value. If it is not, then the argument will be tested to determine if it is a design object. Consequently, where there is ambiguity between a constant string and the name of a design object, the constant will be given precedence. If in this case the design object is desired, the full hierarchical path to the object can be supplied to differentiate it from the constant string.

Examples

Calling a System Verilog Static or Class Function

- Call using a static declaration path, where the function `sf_voidstring()` is a static class function that accepts a string:

```
call sim:/user_pkg::myfcns::sf_voidstring first_string
```

- Call using a class instance path to specify the function, where the function f_intint() of the class type /utop/tmyfcns accepts an integer:

```
call /utop/tmyfcns.f_intint 37
```

- Call using a class instance path to specify the function, and pass in a class instance (/utop/tmyfcns is a class handle):

```
call /utop/tmyfcns.f_voidclasscolor /utop/tmyfcns
```

- Call using a class instance path, and pass in a class instance as an argument using a class instance id string

```
call /utop/tmyfcns.f_voidclasscolor @myType@3
```

- Call using a class instance id string to specify the function:

```
call @myType@543.get_full_name
```

- Call using a declaration path, where the function is non-static so a class instance must also be supplied. The member function f_voidstring() accepts a string:

```
call sim:/user_pkg::myfcns::f_voidstring /my/class/instance "some string"
```

- Call using a class instance id string to specify the function where the function returns a string:

```
call @uvm_sequencer_3@3.get_full_name
```

Returns:

```
# test.e2_a.sequencer
```

- Call using a relative class hierarchical name to specify the function where the function returns a class handle:

```
call moduleX.who_am_i
```

Returns:

```
# @myClassX@4
```

Calling a System Task

- Call \$display with literal values:

```
call \$display {"%0s"} {"Hello from TCL!"}
```

Returns:

```
# Hello from TCL!
```

```
call -builtin display {"%0d"} 'd2999
```

Returns:

```
# 2999
```

```
call -usertf display {"%0d"} 'd3999
```

Returns:

```
# ** Error: Expected user-defined system task $display not found in
the context (/top2).
```

- Call \$display with literal values:

```
call \$display {"top2.i=%0d top2.r=%0b"} top2/i top2/r
```

Returns:

```
# top2.i=5 top2.r=110
```

Calling a System Function

In the following examples \$pow is a user defined function that raises the 1st argument to the power of the 2nd (for example, \$pow(a, b) => ab)

- Call \$pow with literal values:

```
call \$pow 2 1
```

Returns:

```
# 2
```

```
call \$pow 3 2
```

Returns:

```
# 9
```

```
call \$pow [call \$pow 2 1] [call \$pow 3 2]
```

Returns:

```
# 512
```

- Call \$pow with variable values:

```
call -env /top/u1 display r1
```

Returns:

```
# 7
```

```
call -env /top/u2 display r2
```

Returns:

```
# 2
```

```
call pow /top/u1/r1 /top/u2/r2
```

Returns:

```
# 49
```

cd

Changes the ModelSim local directory to the specified directory.

Syntax

cd [[<dir>](#)]

Arguments

- <dir>

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

Description

Executing a cd command closes the current project. You cannot execute this command while a simulation is in progress.

change

This command modifies the value of a: VHDL constant, generic, or variable; Verilog register or variable

Syntax

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

Description

For VHDL constants, the change command may not affect all uses of deferred (or other) constants, because the simulation depends on constants not changing after elaboration. You must treat VHDL constants as immutable after compilation.

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.

Supported objects include:

- VHDL

- Scalar variable, constant, and 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 sub-element 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.
- Sub-elements 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.

- <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 (""). If the string contains spaces, the quoted string must be placed inside curly braces ({}).

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 does 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 (for example, 16).
change {rega[16]} 0
- Change the value of the set of elements of foo that is specified by the slice (for example, 20:22).
change {foo[20:22]} 011
- 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 in curly braces because of the space between the value and the units.
change mytimegeneric {"500 ps"}

classinfo ancestry

Returns class inheritance hierarchy for a named class type.

Syntax

```
classinfo ancestry [-dataset <name>] [-n] [-o <outfile>] [-tcl] <class_type>
```

Arguments

- <class_type>
(required) Name of the class type or the full path to the class type.
- -dataset <name>
(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.
 <name> — The name of an open dataset.
- -n
(optional) Returns class type names only. Does not include the path unless required to resolve name ambiguity.
- -o <outfile>
(optional) Sends the results of the command to <outfile> instead of the transcript.
 <outfile> — Specifies the name of the file to write output to.
- -tcl
(optional) Returns a tcl list instead of formatted output.

Related Topics

[ClassDebug \[ModelSim User's Manual\]](#)

[classinfo descriptive](#)

[classinfo find](#)

[classinfo instances](#)

[classinfo isa](#)

[classinfo report](#)

[classinfo stats](#)

[classinfo trace](#)

[classinfo types](#)

classinfo descriptive

Returns the descriptive class name for the specified authoritative class name.

Syntax

```
classinfo descriptive [-dataset <name>] [-exact | -glob | -regexp] [-tcl] [-o <outfile>]  
                      <class_type>
```

Arguments

- <class_type>
(required) Treats <class_type> as a glob-style expression and returns all matches to the transcript. Wildcard characters asterisk (*) and question mark (?) are permitted.
- -dataset <name>
(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.
 <name> — The name of an open dataset.
- -exact
(optional) Returns results that match <class_type> exactly.
- -glob
(optional, default) Treats <class_type> as a glob-style expression. Wildcard characters asterisk (*) and question mark (?) are permitted.
- -o <outfile>
(optional) Sends the results of the command to <outfile> instead of the transcript.
 <outfile> — Specifies the name of the file to write the output to.
- -regexp
(optional) Treats <class_name> as a regular expression.
- -tcl
(optional) Returns a tcl list instead of formatted output.

Related Topics

- [ClassDebug \[ModelSim User's Manual\]](#)
- [Logging Class Types and Class Instances \[ModelSim User's Manual\]](#)
- [Working with Class Types \[ModelSim User's Manual\]](#)
- [Analyzing Class Types \[ModelSim User's Manual\]](#)
- [classinfo ancestry](#)
- [classinfo find](#)

[classinfo instances](#)

[classinfo isa](#)

[classinfo report](#)

[classinfo stats](#)

[classinfo trace](#)

[classinfo types](#)

classinfo find

Reports on the current state of a specified class instance; whether it exists, has not yet been created, or has been destroyed.

Syntax

```
classinfo find [-dataset <name>][-tcl] [-o <outfile>] <class_instance_identifier>
```

Arguments

- <class_instance_identifier>
(required) Class instance identifier of the specific class instance to find.
- -dataset <name>
(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.
 <name> — The name of an open dataset.
- -o <outfile>
(optional) Sends the results of the command to <outfile> instead of the transcript.
 <outfile> — Specifies the name of the file to write the output to.
- -tcl
(optional) Returns a tcl list instead of formatted output.

Examples

- Find the class instance @mem_item@87

```
VSIM> classinfo find @mem_item@87
```

Returns:

```
# @mem_item@87 has been destroyed
```

- Find the class instance @mem_item@200

```
VSIM> classinfo find @mem_item@200
```

Returns:

```
# @mem_item@200 not yet created
```

Related Topics

[ClassDebug \[ModelSim User's Manual\]](#)

[classinfo ancestry](#)

[classinfo descriptive](#)

[classinfo instances](#)

[classinfo isa](#)
[classinfo report](#)
[classinfo stats](#)
[classinfo trace](#)
[classinfo types](#)

classinfo implements

Displays a list of the classes which implement a specified SystemVerilog interface class. The type of the class argument affects the contents of the list.

Syntax

```
classinfo implements [-dataset <name>] [-tcl] [-o <outfile>] <class_type>
```

Arguments

- <class_type>

(required) Name of the SystemVerilog class to use to generate the output listing. The type of this class determines the type of classes listed, as follows:

- If <class_type> is not an interface class, the output indicates which interface classes that class implements.
- If <class_type> is an interface class, the output indicates which classes implement that interface class.

- -dataset <name>

(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.

 <name> — The name of an open dataset.

- -o <outfile>

(optional) Sends the results of the command to <outfile> instead of the transcript.

 <outfile> — Specifies the name of the file where the output will be written.

- -tcl

(optional) Returns a tcl list instead of formatted output.

Examples

The following module defines classes labeled A, B, C1, C2, M, N, X, Y:

```
module test8;
    interface class M; endclass
    interface class N; endclass
    interface class X extends M, N; endclass
    interface class Y extends M; endclass
    class A implements M; endclass
    class B extends A implements X; endclass
    class C1 extends B implements Y; endclass
    class C2 extends B; endclass
endmodule
```

- Use interface class M as argument:

```
vsim> classinfo implements M
```

Output list:

```
# /test8/A implements /test8/M
# /test8/B implements /test8/M
# /test8/C1 implements /test8/M
# /test8/C2 implements /test8/M
```

- Use class A as argument:

vsim> classinfo implements A

Output list:

```
# /test8/A implements /test8/M
```

- Use class B as argument to access extended classes defined in test8:

vsim> classinfo implements B

Output list:

```
# /test8/B implements /test8/M
# /test8/B implements /test8/N
# /test8/B implements /test8/X
```

Related Topics

[ClassDebug \[ModelSim User's Manual\]](#)

[classinfo descriptive](#)

[classinfo find](#)

[classinfo instances](#)

[classinfo interfaces](#)

[classinfo isa](#)

[classinfo report](#)

[classinfo stats](#)

[classinfo trace](#)

[classinfo types](#)

classinfo instances

Reports the list of existing class instances of a specific class type. You can use this to determine the class instances to log or examine, and to debug excessive memory use problems caused by class instances that are not cleaned up properly.

Syntax

```
classinfo instances [-dataset <name>] [-tcl] [-verbose] [-o <outfile>] <class_type>
```

Arguments

- **<class_type>**
(required) Name of the class type or the full path of the class type. If this is an interface class, the output lists all instances that implement that interface class.
- **-dataset <name>**
(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.
 <name> — The name of an open dataset.
- **-tcl**
(optional) Returns a tcl list instead of formatted output.
- **-verbose**
(optional) Includes the classname in the output along with the instance name.
- **-o <outfile>**
(optional) Sends the results of the command to <outfile> instead of the transcript.
 <outfile> — Specifies the name of the file to write the output to.

Examples

- List the current instances for the class type mem_item.

```
vsim> classinfo instances mem_item
```

Returns:

```
# @mem_item@140
# @mem_item@139
# @mem_item@138
# @mem_item@80
# @mem_item@76
# @mem_item@72
# @mem_item@68
# @mem_item@64
```

- The following module defines classes labeled A, B, C1, C2, M, N, X, Y:

```
module test8;
    interface class M; endclass
    interface class N; endclass
    interface class X extends M, N; endclass
    interface class Y extends M; endclass
    class A implements M; endclass
    class B extends A implements X; endclass
    class C1 extends B implements Y; endclass
    class C2 extends B; endclass
endmodule
```

The following commands show the difference between using and omitting the -verbose argument.

vsim> classinfo instances -verbose M

Returns:

```
# @A@1 /test8/A
# @B@1 /test8/B
```

vsim> classinfo instances -verbose A

Returns:

```
# @A@1 /test8/A
```

vsim> classinfo instances M

Returns:

```
# @A@1
# @B@1
```

vsim> classinfo instances A

Returns:

```
# @A@1
```

Related Topics

[ClassDebug \[ModelSim User's Manual\]](#)

[classinfo ancestry](#)

[classinfo descriptive](#)

[classinfo find](#)

[classinfo implements](#)

[classinfo interfaces](#)

[classinfo isa](#)

[classinfo report](#)

classinfo instances

[classinfo stats](#)

[classinfo trace](#)

[classinfo types](#)

classinfo interfaces

Lists the interface class types that match a specified pattern. Finds all interface classes that match a regular expression and determines the full path of interface class types.

Syntax

```
classinfo interfaces [-dataset <name>] [-tcl] [-o <outfile>] [<class_type>]
```

Arguments

- <class_type>
(optional) Name of the interface class type or the full path to the interface class type. If omitted, all interface classes are listed.
- -dataset <name>
(optional) Specifies an open dataset to search for interface class information. The default is to search the currently active dataset.
 <name> — The name of an open dataset.
- -o <outfile>
(optional) Sends the results of the command to <outfile> instead of the transcript.
 <outfile> — Specifies the name of the file to write the output to.
- -tcl
(optional) Returns a tcl list instead of formatted output.

Examples

The following module defines classes labeled A, B, C1, C2, M, N, X, Y:

```
module test8;
    interface class M; endclass
    interface class N; endclass
    interface class X extends M, N; endclass
    interface class Y extends M; endclass
    class A implements M; endclass
    class B extends A implements X; endclass
    class C1 extends B implements Y; endclass
    class C2 extends B; endclass
endmodule
```

- Used with no argument, the command returns the names of all interface classes:

```
vsim> classinfo interfaces
```

Output list:

```
# /test8/M
# /test8/N
# /test8/X
# /test8/Y
```

Related Topics

[ClassDebug \[ModelSim User's Manual\]](#)

[classinfo descriptive](#)

[classinfo find](#)

[classinfo implements](#)

[classinfo instances](#)

[classinfo isa](#)

[classinfo report](#)

[classinfo stats](#)

[classinfo trace](#)

[classinfo types](#)

classinfo isa

Returns to the transcript a list of all classes extended from the specified class type.

Syntax

```
classinfo isa [-dataset <name>] [-n] [-o <outfile>] [-tcl] <class_type>
```

Arguments

- <class_type>
(required) Name of the class type or the full path of the class type.
- -dataset <name>
(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.
 <name> — The name of an open dataset.
- -n
(optional) Returns class names only. Does not include the path unless required to resolve name ambiguity.
- -o <outfile>
(optional) Sends the results of the command to <outfile> instead of the transcript.
 <outfile> — Specifies the name of the file to write the output to.
- -tcl
(optional) Returns a tcl list instead of formatted output.

Related Topics

[ClassDebug \[ModelSim User's Manual\]](#)

[classinfo ancestry](#)

[classinfo descriptive](#)

[classinfo find](#)

[classinfo instances](#)

[classinfo report](#)

[classinfo stats](#)

[classinfo trace](#)

[classinfo types](#)

classinfo report

Prints detailed reports on class instance usage. Displays columns for class type names and their current, peak and total class instance counts.

Syntax

```
classinfo report [-c [fntpc]] [-dataset <name>] [-m <maxout>] [-o <outfile>]  
[-sort [a | d] [f | n | t | p | c]] [-tcl] [-z]
```

Arguments

- **-c [fntpc]**
(optional) Display the report columns in the specified order in a report. The default is to display all columns in the following order: Full Path, Class Name, Total, Peak, Current. You can specify one or more columns in any order.
 - f — The Full Path column displays the full path name.
 - n — The Class Name column displays the class name, and for parameterized classes, it displays the internally generated class specialization name.
 - t — The Total column displays the total number of instances of the named class.
 - p — The Peak column displays the maximum number of instances of the named class that existed simultaneously at any time in the simulation.
 - c — The Current column displays the current number of instances of the named class.
- **-dataset <name>**
(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.
 - <name> — The name of an open dataset.
- **-m <maxout>**
(optional) Display the specified number of lines of the report.
 - <maxout> — Any non-negative integer.
- **-o <outfile>**
(optional) Sends the results of the command to <outfile> instead of to the transcript.
 - <outfile> — Specifies the name of the file to write the output to.
- **-sort [a | d] [f | n | t | p | c]**
(optional) Specifies whether to sort report information in ascending or descending order, and which column to sort by. You can specify only one column for sorting.
 - a — Sort the entries in ascending order.
 - d — Sort the entries in descending order.
 - f — Sort by the Full Path column

- n — Sort by the Class Name column
t — Sort by the Total column
p — Sort by the Peak column
c — Sort by the Current column
- -tcl
 - (optional) Returns a tcl list instead of formatted output.
 - -z
 - (optional) Remove all items from the report that have a total instance count of zero.

Examples

- Create a report of all class instances in descending order in the Total column. Print the Class Names, Total, Peak, and Current columns. List only the first six lines of the report.

```
vsim> classinfo report -s dt -c ntpc -m 6
```

Returns:

# Class Name	Total	Peak	Current
# uvm_pool_11	318	315	315
# uvm_event	286	55	52
# uvm_callback_iter_1	273	3	2
# uvm_queue_3	197	13	10
# uvm_object_string_pool_1	175	60	58
# mem_item	140	25	23

Related Topics

- [ClassDebug \[ModelSim User's Manual\]](#)
[classinfo ancestry](#)
[classinfo descriptive](#)
[classinfo find](#)
[classinfo instances](#)
[classinfo isa](#)
[classinfo stats](#)
[classinfo trace](#)
[classinfo types](#)

classinfo stats

Prints statistics about the total number of class types and total, peak, and current class instance counts during the simulation.

Syntax

```
classinfo stats [-dataset <name>] [-tcl] [-o <outfile>]
```

Arguments

- **-dataset <name>**
(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.
 <name> — The name of an open dataset.
- **-o <outfile>**
(optional) Sends the results of the command to <outfile> instead of the transcript.
 <outfile> — Specifies the name of the file to write the output to.
- **-tcl**
(optional) Returns a tcl list instead of formatted output.

Examples

- Display the current number of class types, the maximum number, peak number and current number of all class instances.

```
vsim> classinfo stats
```

Returns:

```
# class type count          451
# class instance count (total) 2070
# class instance count (peak) 1075
# class instance count (current) 1058
```

Related Topics

[ClassDebug \[ModelSim User's Manual\]](#)

[classinfo ancestry](#)

[classinfo descriptive](#)

[classinfo find](#)

[classinfo instances](#)

[classinfo isa](#)

[classinfo report](#)

[classinfo trace](#)

[classinfo types](#)

classinfo trace

Displays the active references to the specified class instance. Useful in debugging situations where class instances are not being destroyed as expected because something in the design is still referencing them. Finding those references can lead to uncovering bugs in class reference management, and lead to large memory savings.

Syntax

```
classinfo trace [-dataset <name>] [-m <maxout>] [-tcl] [-o <outfile>] <class_instance_name>
```

Arguments

- <class_instance_name>
(required) Name of the class item in the following format @<name>@#.
- -dataset <name>
(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.
 <name> — The name of an open dataset.
- -m <maxout>
(optional) Display the specified number of lines of the report.
 <maxout> — Any non-negative integer.
- -o <outfile>
(optional) Sends the results of the command to <outfile> instead of the transcript.
 <outfile> — Specifies the name of the file to write the output to.
- -tcl
(optional) Returns a tcl list instead of formatted output.

Examples

- Report the active references to @mem_item@200

```
VSIM> classinfo trace @uvm_resource_14@2
```

Returns:

```
#{uvm_pkg::uvm_resources.rtab["mem_interface"].queue[15] }  
#{uvm_pkg::uvm_config_db::uvm_config_db__12::m_rsc[@uvm_root@1].  
pool["uvm_test_topmem_interface"] }
```

Related Topics

[ClassDebug \[ModelSim User's Manual\]](#)

[classinfo ancestry](#)

[classinfo descriptive](#)

[classinfo find](#)

[classinfo instances](#)

[classinfo isa](#)

[classinfo report](#)

[classinfo stats](#)

[classinfo types](#)

classinfo types

Lists the class types that match or do not match a specified pattern. Finds all classes that match a regular expression and determines the full path of class types.

Syntax

```
classinfo types [-dataset <name>] [-exact | -glob | -regexp] [-n] [-o <outfile>] [-tcl] [-x]
    <pattern>
```

Arguments

- <pattern>
(required) A standard TCL glob expression used as a search string.
- -dataset <name>
(optional) Specifies an open dataset to search for class information. The default is to search the currently active dataset.
 <name> — The name of an open dataset.
- -exact
(optional) Returns results that match <pattern> exactly.
- -glob
(optional) Returns glob styles matches for <pattern>.
- -n
(optional) Returns class names only. Does not include the path unless required to resolve name ambiguity.
- -o <outfile>
(optional) Sends the results of the command to <outfile> instead of the transcript.
 <outfile> — Specifies the name of the file to write the output to.
- -regexp
(optional) Returns regular expressions that match <pattern>.
- -tcl
(optional) Returns a tcl list instead of formatted output.
- -x
(optional) Display classes that do not match the pattern.

Examples

- List the full path of the class types that do not match the pattern *uvm*.

```
vsim> classinfo types -x *uvm*
```

Returns:

```
# /environment_pkg::test_predictor
# /environment_pkg::threaded_scoreboard
# /mem_agent_pkg::mem_agent
# /mem_agent_pkg::mem_config
# /mem_agent_pkg::mem_driver
```

Related Topics

[ClassDebug \[ModelSim User's Manual\]](#)

[classinfo ancestry](#)

[classinfo descriptive](#)

[classinfo find](#)

[classinfo implements](#)

[classinfo instances](#)

[classinfo interfaces](#)

[classinfo isa](#)

[classinfo report](#)

[classinfo stats](#)

[classinfo trace](#)

configure

Invokes the List or Wave widget configure command for the current default List or Wave window.

Syntax

Base Command Usage

```
configure list | wave [<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>]]] [-namecwidth [<width>]] [-rowmargin [<pixels>]]  
[-signalnamewidth [<value>]] [-timecolor [<color>]] [-timeline [0 | 1]]  
[-timelineunits [fs | ps | ns | us | ms | sec | min | hr]] [-valuecwidth [<width>]]  
[-vectorcolor [<color>]] [-waveselectcolor [<color>]] [-waveselectenable [0 | 1]]
```

Description

The command works in three modes:

- With no 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 includes 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 the first argument to the configure command.
 - list — Specifies the List widget.
 - wave — Specifies the Wave widget.
 - <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 displays 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 length of time gating remains 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 is displayed after the gate expression becomes false (if you do not 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.
- -usegating [off | on]
 - (optional) Enables triggers to be gated on or off by an overriding expression. (Use the -gateexpr argument to specify the expression.) Refer to “[Using Gating Expressions to Control Triggering](#)” in the GUI Reference Manual 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. The default is to specify the current simulation resolution by omitting <unit>. If you specify <unit>, you must enclose <delay> and <unit> within curly braces ({}). Valid time units are: fs, ps, ns, us, ms, sec, min, and hr.
- **-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. The default is to specify the current simulation resolution by omitting <unit>. If you specify <unit>, you must enclose <delay> and <unit> within curly braces ({}). Valid time units are: fs, ps, ns, us, ms, sec, min, and hr.
- **-usesignaltriggers [0 | 1]**
(optional) Specifies whether or not to use signals as triggers.
 0 — Signals are not used as triggers
 1 — Signals are used as triggers
- **-usestrobe [0 | 1]**
(optional) Specifies whether or not to use a strobe 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. The default is 2.
- **-cursorlockcolor [<color>]**
(optional) Specifies the color of a locked cursor. Related Tcl variable is PrefWave(cursorLockColor).
 <color> — Any Tcl color. The default is red.
- **-gridauto [off | on]**
(optional) Controls the grid period when in simulation time mode.
 off — (default) Uses a user-specified grid period.
 on — The major tick marks in the time line determine the grid period.

- **-gridcolor [<color>]**
(optional) Specifies the background grid color. Related Tcl variable is PrefWave(gridColor).
 <color> — Any color. The default is grey50.
- **-griddelta [<pixels>]**
(optional) Specifies how close (in pixels) two grid lines can be drawn before intermediate lines are removed. Related Tcl variable is PrefWave(gridDelta).
 <pixels> — Any non-negative integer. 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. The default is 0.
 <unit> — (optional) A suffix specifying a unit of time. The default is to specify the current simulation resolution by omitting <unit>. If you specify <unit>, you must enclose <delay> and <unit> within curly braces ({}). Valid time units are: fs, ps, ns, us, ms, sec, min, and hr.
- **-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. The default is 1.
 <unit> — (optional) A suffix specifying a unit of time. The default is to specify the current simulation resolution by omitting <unit>. If you specify <unit>, you must enclose <delay> and <unit> within curly braces ({}). Valid time units are: fs, ps, ns, us, ms, sec, min, and hr.
- **-namecolwidth [<width>]**
(optional) Specifies the width of the name column in pixels. Related Tcl variable is PrefWave(nameColWidth).
 <width> — Any non-negative integer. 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. 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. 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. 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) Display simulation time.
 1 — Display grid period count.
- **-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. The default is 100.
- **-vectorcolor [<color>]**
(optional) Specifies the vector waveform color. Default is #b3ffb3. Related Tcl variable is PrefWave(vectorColor).
 <color> — Any color. 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. 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.

Examples

- Display the current value of the strobeperiod attribute.

config list -strobeperiod

- Set the period of the list strobe and turns it on.

config list -strobeperiod {50 ns} -strobestart 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

[Setting GUI Preferences \[ModelSim GUI Reference Manual\]](#)

dataset alias

Maps an alternate name (alias) to an open dataset.

Syntax

dataset alias <dataset_name> [<alias_name>]

Description

A dataset can have any number of unique aliases, but all dataset names and aliases must be still be unique when more than one dataset is open. Aliases are not saved to the .wlf file, and you must remap them if you close and re-open the dataset.

Arguments

- <dataset_name>
(required) Specifies a dataset name or currently assigned dataset alias. Must be the first argument to the dataset alias command. Returns a list of all aliases mapped to the specified dataset file when specified without <alias_name>.
- <alias_name>
(optional) Specifies string to assign to the dataset as an alias. Allows wildcard characters.

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

Removes all event data from the current simulation WLF file, while retaining all currently logged signals. Subsequent run commands continue to accumulate data in the WLF file.

Note

 This command applies only to WLF-based simulation datasets.

Syntax

dataset clear

Description

Running this command with no design loaded returns the error “Dataset not found:sim”. If you run the command with a design loaded, the “sim:” dataset is cleared, regardless of which dataset is currently set. Clearing the dataset clears any open Wave window based on the “sim:” dataset.

Arguments

None

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 \[ModelSim User's Manual\]](#)

dataset close

Closes an active dataset.

Note

 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 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

Configures WLF parameters for an open dataset and all aliases mapped to that dataset.

Syntax

```
dataset config <dataset_name> [-wlfcachesize [<n>]] [-wlfdeleteonquit [0 | 1]] [-wlfopt [0 | 1]]
```

Arguments

- <dataset_name>
(required) Specifies an open dataset or dataset alias to configure. Must be the first argument to the dataset config command.
- -wlfcachesize [<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.
- -wlfdeleteonquit [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.
- -wlfopt [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 -wlfcachesize 512
```

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 \[ModelSim User's Manual\]](#)

dataset current

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 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 \[ModelSim User's Manual\]](#)

dataset info

Reports a variety of information about a dataset.

Syntax

```
dataset info {name | file | exists} <dataset_name>
```

Arguments

- {name | file | exists}

(required) Identifies what type of information to report.

Allows only one option per command. 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 file associated with the dataset.

exists — Returns "1" if the dataset is currently open, "0" if it is not.

Must be 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

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

Opens a WLF file (either the currently running *vsim.wlf* or a saved WLF file), and assigns it the logical name you specify.

Syntax

```
dataset open <file_name> [<dataset_name>]
```

Description

The file can be the existing WLF file for a currently running simulation. To close a dataset, use the [dataset close](#) command.

Arguments

- **<file_name>**
(required) Specifies the file to open as a view-mode dataset. Must be the first argument to the dataset open command. Specify *vsim.wlf* to open the currently running WLF file.
- **<dataset_name>**
(optional) Specifies a name for the open dataset. This name identifies the dataset in the current session. By default the dataset prefix is the name of the specified 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](#)

dataset rename

Changes the name of a dataset to the new name you specify.

Syntax

```
dataset rename <dataset_name> <new_dataset_name>
```

Arguments

- <dataset_name>
Specifies the existing name of the dataset. Must be the first argument.
- <new_dataset_name>
Specifies the new name for the dataset. Must be the second argument.

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

Unloads the specified dataset or currently active dataset and reloads the dataset using the same dataset name.

Syntax

dataset restart [<file_name>]

Arguments

- <file_name>

(optional) Specifies the file to open as a dataset. If <filename> is not specified, the currently active dataset restarts.

Related Topics

[dataset alias](#)

[dataset clear](#)

[dataset close](#)

[dataset config](#)

[dataset info](#)

[dataset list](#)

[dataset open](#)

[dataset rename](#)

[dataset save](#)

[dataset snapshot](#)

dataset save

Writes data from the current simulation to a specified file while the simulation is in progress.

Syntax

dataset save <dataset_name> <file_name>

Arguments

- <dataset_name>
(required) Specifies the name of the dataset you want to save. Must be the first argument.
- <file_name>
(required) Specifies the name of the file to save. Must be the second argument.

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

Saves data from the current WLF file (*vsim.wlf* by default) at a specified interval. Provides you with sequential or cumulative "snapshots" of your simulation data.

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 to save the files. Either absolute or relative paths can be used. Default is to save to the current working directory.
- **-disable**
(optional) Turns snapshotting off. Stores all dataset snapshot settings from the current simulation 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; the default is *vsim_snapshot.wlf*. The suffix *.wlf* is appended to the specified filename. Some arguments may also append 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 — Creates a new file for each snapshot. Adds an incrementing suffix (1 to n) 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 — Clears the current WLF file every time a snapshot is taken.

- **-report**
(optional) Lists current snapshot settings in the Transcript window and ignores all other options.
- **-reset**
(optional) Resets values back to defaults, then applies the remainder of the arguments on the command line. See Examples section, below. If specified 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 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 the final argument to the dataset snapshot command.
 <n> — Any positive integer.
 <unit> — (optional) A suffix specifying a unit of time. The default is to specify the current simulation resolution by omitting <unit>. If you specify <unit>, you must enclose <limit> and <unit> within braces ({}). Valid time units are: fs, ps, ns, us, ms, sec, min, and hr.

Examples

- Create the file *vsim_snapshot_<n>.wlf*, and write to it every time the current WLF file reaches a multiple of 10 MB (that is, at 10 MB, 20 MB, 30 MB, and so on.).
dataset snapshot -size 10
- Similar to the previous example, but in this case, clear the current WLF file 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 1000000 to 2000000, and *gold_3.wlf* with data from 2000000 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 3000000 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

Removes objects from either the List or Wave window.

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.) Uses the default window, if none is specified. The default window is determined by the most recent invocation of the view command and has “-Default” appended to the name.
- `<object_name>...`
(required) Specifies the name of an object. Must match an object name used in an `add list` or `add wave` command. Specify multiple object names as a space-separated list. Allows wildcard characters. Must be 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*`

describe

Displays information about simulation objects and design regions in the Transcript window.

Syntax

describe <name>...

Description

This command displays information about the following types of simulation objects and design regions:

- VHDL — signals, variables, constants, and FILE objects.
- Verilog — nets and registers
- Design region

VHDL signals, Verilog nets and registers, can be specified as hierarchical names.

Arguments

- <name>...

(required) The name of an HDL object for which you want a description.

Specify multiple object names as a space separated list. Allows wildcard characters. HDL object names can be relative or full hierarchical names.

Examples

- Print the types of the three specified signals.

describe clk prw prdy

- Return information about /textio/INPUT.

describe /textio/INPUT

produces:

```
# File of
#   Unconstrained Array of
#     VHDL standard type CHARACTER
```

disablebp

Turns off breakpoints and when commands.

Note

 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.

- <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

[enablebp](#)

[onbreak](#)

do

Executes the commands contained in a DO file.

Syntax

do <filename> [<parameter_value>...]

Description

A DO file can have any name and extension. Any encountered errors interrupt the DO file script execution, unless you specify a [onerror](#) command, or the OnErrorDefaultAction Tcl variable with the [resume](#) command. You can use the [onbreak](#) command to take action with source code breakpoint cases.

Arguments

- <filename>

(required) Specifies the name of the DO 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 the first argument to the do command.

If the do command is executed from another DO file, pathnames are relative to the directory of the calling DO file. This allows groups of DO 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 DO file. You must separate multiple parameter values 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 file), you must use the argc simulator state variable in the DO file script. Refer to “[Simulator State Variables](#)” and “[Making Script Parameters Optional](#)” in the User’s Manual.

Note

 There is no limit on the number of parameters that can be passed to macros, but 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 DO file.

do macros/stimulus 100

Where the DO 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 DO Files \[ModelSim User's Manual\]](#)
- [General Modes of Operation \[ModelSim User's Manual\]](#)
- [Using a Startup File \[ModelSim User's Manual\]](#)
- [DOPATH \[ModelSim User's Manual\]](#)
- [Saving a Transcript File as a DO File \[ModelSim GUI Reference Manual\]](#)

drivers

Displays the names and strength of all drivers of the specified object.

Syntax

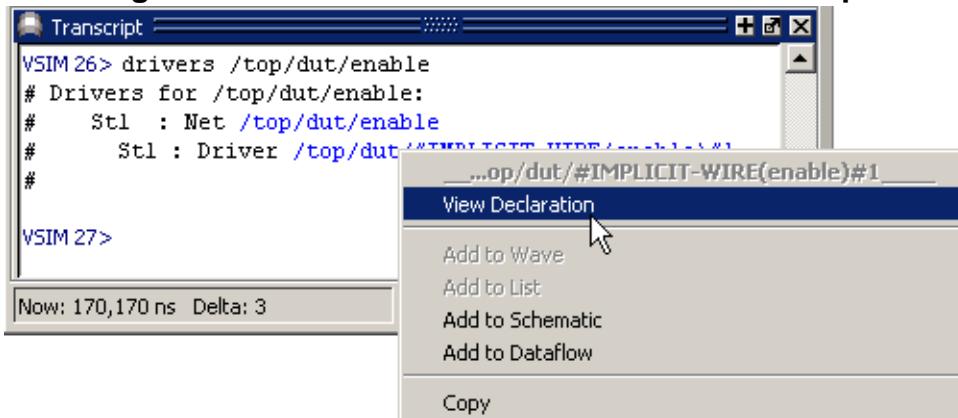
```
drivers <object_name> [-source]
```

Description

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.

The output from the drivers command appears in the Transcript window as a hypertext link, allowing you to right-click to open a drop-down menu and quickly add signals to various windows. It includes a "View Declaration" item to open the source definition of the signal.

Figure 2-1. drivers Command Results in Transcript



Arguments

- `<object_name>`
(required) Specifies the name of the signal or net for which to display drivers. All signal or net types are valid. Accepts multiple names and wildcards.
- `-source`
(optional) Returns the source file name and line number for each driver of the specified signal or net. Returns the value n/a when unable to determine the source location for the driver.

Examples

```
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.

dumplog64

Dumps the contents of the specified WLF file to stdout in a readable format.

Note

 When you use this command, you cannot open the WLF file for writing in a simulation. You cannot use this command in a DO file.

Syntax

`dumplog64 <filename>`

Arguments

- `<filename>`
(required) The name of the WLF file to read.

echo

Displays a specified message in the Transcript window.

Syntax

`echo [<text_string>]`

Arguments

- `<text_string>`

(required) Specifies the message text to display. If you surround the text string with quotation marks, blank spaces display as entered. If you omit quotation marks, adjacent blank spaces are compressed into a single 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 adjacent blank spaces 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

Invokes the editor specified by the EDITOR environment variable. By default, the specified filename opens in the Source window.

Syntax

`edit [<filename>]`

Arguments

- `<filename>`

(optional) Specifies the name of the file to edit. If you omit the `<filename>` argument, the editor opens the current source file. If you specify a non-existent filename, the editor opens a new file. You can use either absolute or relative paths.

Related Topics

[notepad](#)

[EDITOR \[ModelSim User's Manual\]](#)

enablebp

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.
- `<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

[disablebp](#)

[onbreak](#)

encoding

Translates between the 16-bit Unicode characters used in Tcl strings and a named encoding, such as Shift-JIS. Consists of several related commands.

Syntax

```
encoding convertfrom <encoding_name> <string>
encoding convertto <encoding_name> <string>
encoding names
encoding system <encoding_name>
```

Description

The encoding commands work with the encoding of your character representations in the GUI:

- `encoding convertfrom` — Converts a string from the named encoding to Unicode.
- `encoding convertto` — Converts a string to the named encoding from Unicode.
- `encoding names` — Returns a list of all valid encoding names (takes no arguments).
- `encoding system` — Changes the current system encoding to a named encoding. If you omit a new encoding the command returns the current system encoding. The system encoding is used whenever Tcl passes strings to system calls.

Arguments

- `string`
Specifies a string to be converted.
- `encoding_name`
The name of the encoding to use.

environment

Allows you to display or change the current dataset and region/signal environment.

Note

 You can abbreviate this command as “env”, as shown in the Examples section, below.

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 displays the pathname of the current region/signal environment.
- **-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

[Setting your Context by Navigating Source Files \[ModelSim GUI Reference Manual\]](#)

examine

Examines one or more objects and displays current values (or the values at a specified previous time) in the Transcript window.

Note

 You can abbreviate the examine command as “exa”.

Syntax

```
examine <name>... [-delta <delta>] [-env <path>] [-event <time>] [-handle] {[-in] [-out] [-inout] | [-ports]} [-internal] [-expr <expression>] [-name] [-<radix_type>] [-radix <radix_type>][,<radix_flag>][,...]] [-radixenumnumeric | -radixenumsymbolic] [-showbase | -noshowbase] [-time <time>] [-value]
```

Description

The examine command can also compute the value of an expression of one or more objects.

You can examine the following objects:

- VHDL — signals, shared variables, process variables, constants, generics, and FILE objects
- Verilog — nets, registers, parameters, and variables

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 can also be examined within the GUI (actual signals are examined in the kernel).

The examine command uses the following rules 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 are 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, with the result that some objects that may be visible from a given context will not be found when using wildcards 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 never matches a path separator. For example, /dut/* matches /dut/siga and /dut/clk, while /dut* does not match either of those.

If you do not specify a radix with the examine command, the default radix and radix flags are used.

- Set the default radix with the [radix](#) command or by editing the DefaultRadix variable in the modelsim.ini file. Refer to [DefaultRadix](#) in the User's Manual.
- Set the default radix flags value with the radix command or by editing the DefaultRadixFlags variable in the modelsim.ini file. Refer to [DefaultRadixFlags](#) in the User's Manual.
- Specifying examine -<radix_type> returns the value of the object in the specified radix and default radix flags value.
- Specifying examine -radix <radix_type> returns the value of the object in the specified radix.
- Specifying examine -radix [<radix_type>][, [<radix_flag>]] returns the value of the object in the specified radix and radix flags.
- Specifying examine -radix <radix_flag>[,<radix_flag>] returns the value of the object in the default radix and specified radix flags.

For example, assume a default of hexadecimal + showbase:

```
examine d
# 16'h0009

examine -binary d
# 16'b0000000000001001

examine -radix binary d
# 0000000000001001

examine -radix binary,showbase d
# 16'b0000000000001001

examine -radix hex,enumsymbolic nxt_state
# send5

examine -radix hex,enumnumeric nxt_state
```

```
# 0000000d
```

Refer to [Design Object Names](#) for more information on specifying names.

Arguments

- <name>...

(required except when specifying -expr.) Specifies the name of any HDL object.

Allows all object types except those of the type file. Accepts multiple names and wildcards. Spaces, square brackets, and extended identifiers require 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. The default is 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 is 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 does 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.
- -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](#).
- -<radix_type>
 - (optional) Specifies the radix type for the objects that follow in the command. Retains the current flag value for the objects that follow in the command. Valid entries (or any unique abbreviations) are: ascii, binary, decimal, fpoint, hexadecimal, octal, signed, sfixed, symbolic, time, ufixed, unsigned, and default.

This option overrides the global setting of the default radix. (Refer to the description of the modelsim.ini [DefaultRadix](#) variable in the User's Manual).

- -radix [<radix_type>][,<radix_flag>][,...]

(optional) Specifies the radix and/or the radix flags to be used by the examine command. You can use the -radix <radix_type> switch in place of examine -<radix_type>.

<radix_type> — (required unless specifying <radix_flag>) Specifies a radix and clears the radix flags for the objects that follow in the command. Valid values are: ascii, binary, decimal, fpoint, hexadecimal, octal, pretty, sfixed, symbolic, time, ufixed, unsigned, default, and user-defined radix names (refer to the radix define command).

<radix_flag> — (optional) Sets one or more radix flags on the objects that follow in the command. Specify multiple flags as a comma separated list. Must follow -<radix_type> when the two are specified together.

Valid radix flags:

Table 2-3. Radix flag Arguments to the Examine Command

Argument	Description
enumnumeric	Displays Verilogenums as numbers (formatted by the current radix). This overrides the default behavior of always showing enums symbolically.

Table 2-3. Radix flag Arguments to the Examine Command (cont.)

Argument	Description
enumsymbolic	Restores the default behavior of displaying Verilog enums as symbols by reversing the action of the -radixenumnumeric option.
showbase	Displays the number of bits of the vector and the radix used, where: <ul style="list-style-type: none"> • d = decimal • b = binary • h = hexadecimal • a = ASCII • t = time

For example, instead of simply displaying a vector value of “31”, a value of “16’h31” can be displayed to show that the vector is 16 bits wide, with a hexadecimal radix.

This option overrides the global default settings for the radix and the radix flag (Refer to the descriptions of the modelsim.ini [DefaultRadix](#) and [DefaultRadixFlags](#) in the User’s Manual).

- -radixenumnumeric
(optional) Causes Verilog 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 enums as symbols by reversing the action of the -radixenumnumeric option.
- -showbase | -noshowbase
(optional) Enables or disables displaying the number of bits of the vector and the radix used (binary = b, decimal = d, hexadecimal = h, ASCII = a, and time = t).

For example, with -showbase, instead of simply displaying a vector value of “31”, a value of “16’h31” may be displayed to show that the vector is 16 bits wide, with a hexadecimal radix.

- -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 enclosed in 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 is 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 (16). Note that you must use braces when examining subelements.

examine {rega[16]}

- Return information about `/textio/OUTPUT`

examine /textio/OUTPUT

returns the following:

```
# {STD_OUTPUT {stdout NOTPIPE} WRITE_MODE N/A}
```

The output is a Tcl list with up to four elements. There are three scenarios of results:

- If the file has not been elaborated, the result is the following one-element list:

```
# {"NOT ELABORATED"}
```

- If the file is closed, the result is the following one-element list.

```
# {"CLOSED"}
```

- In all other cases, the result is a four-element list, following the format:

```
# {<file_path> { <descriptor> PIPE | NOTPIPE } <file_mode>
  <file_position>}
```

where,

- `<file_path>` — references either the FILE declaration or corresponding `file_open()` call
- `<descriptor>` — one of the following:
 - `N` — operating system file descriptor resource number.
 - `stdin` — identifying the file as `stdin`.
 - `stdout` — identifying the file as `stdout`.
 - `<file_mode>` — identifying the file mode in which the file was opened, as defined in `std.standard` package; either `READ_MODE`, `WRITE_MODE`, or `APPEND_MODE`.
 - `<file_position>` — value in bytes. For `stdin` or `stdout` files, the value will be

“N/A”.

- Return the value of the contiguous subelements of foo specified by the slice (that is, 20:22). Note the use of 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 between the closing '\ and 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. Quotation marks (" ") are required when spaces are involved; using quotation marks instead of braces causes the Tcl interpreter to expand variables before calling the command.

examine -time \$t -name \$fifo "\${fifo}(1 to 3)" \${fifo}(1)

- Because -time is not specified, this expression is 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 (a blank-separated list of strings). You can enter commands such as:

```
foreach sig [find sig ABC*] {echo "Signal $sig is [exa $sig]" ...}  
if {[examine -bin signal_12] == "11101111XXXZ"} {...}  
examine -hex [find *]
```

Related Topics

[DefaultRadix \[ModelSim User's Manual\]](#)

exit

Exits the simulator and the ModelSim application.

Syntax

```
exit [-force] [-code <integer>]
```

Description

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.

Arguments

- `-force`
(optional) Quits without asking for confirmation. If you omit this argument, 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 "[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. You can use the [onbreak](#) command to specify commands to execute 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.

find

Locates objects by type and name.

Note



Arguments to the command are grouped by object type.

Syntax

```
find nets | signals <object_name> ... [-internal] [-nofilter] {[-in] [-inout] [-out] | [-ports]} [-recursive]  
find instances | blocks {<object_name> ... | -bydu <design_unit> [-filter [<filter_list> | none]] | -file <file_name>} [-blocks | -instances] [-arch] [-protected] [-recursive] [-nodu]  
find virtuals <object_name> ... [-kind <kind>] [-unsaved] [-recursive]  
find classes [<class_name>]  
find objects [-class <class_name>] [-isa <class_name>] [<object_name>]
```

Description

The find command uses the following rules to locate an object:

- If the name does not include a dataset name, use the current dataset.
- If the name does not start with a path separator, use the current context.
- If the name is a path separator followed by a name that is not the name of a top-level design unit, use the first top-level design unit in the design.
- For a relative name containing a hierarchical path, if the first object name cannot be found in the current context, then do an upward search 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 do an upward search 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, if you use wildcards, objects that are visible from a given context, but are within a higher enclosing scope, may not be found.
- 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 never matches a path separator. For example, /dut/* matches /dut/siga and /dut/clk, but /dut* does not.
- Because square brackets are wildcards in the find command, only parentheses (()) can be used to index or slice arrays.

- The find command uses the WildcardFilter Tcl preference variable to exclude the specified types of objects when performing the search.

See [Design Object Names](#) for more information on specifying names.

Arguments

Arguments to the command are grouped by object type.

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.

Note

 The find nets and find signals commands are also supported in vsim -batch mode.

- <object_name> ...
(required) Specifies the net or signal for which you want to search. Allows multiple nets and signals and wildcard characters. 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 includes ports of mode IN.
- -inout
(optional) Specifies that the scope of the search includes ports of mode INOUT.
- -internal
(optional) Specifies that the scope of the search includes internal (non-port) objects.
- -nofilter
(optional) Specifies to ignore the WildcardFilter Tcl preference variable when finding signals or nets.
- -out
(optional) Specifies that the scope of the search includes ports of mode OUT.
- -ports
(optional) Specifies that the scope of the search includes 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.

Note

 The find instances command is also supported in vsim -batch mode.

- **-arch**

Used with “instances” only: Lists the corresponding architecture name, along with the entity name for any VHDL design unit names returned by the find command.

- **-blocks | -instances**

Returns either blocks or instances in addition to the information returned by the base command. For example:

```
find instances -blocks -recursive /test/*
find blocks -instances -recursive /test/*
```

- **-bydu <design_unit> [-filter [<filter_list> | none]]**

Searches for a design unit. Mutually exclusive with -file and <object_name>.

<design_unit>

Name of a single design unit to search for. This argument matches the pattern specified by primary <design_unit> of the instance only. Library and Secondary names are not supported.

-filter <filter_list>

(applies only to find instances) Removes information returned by the command to those types defined in the filter_list.

<filter_list>

This is a space separated group of arguments based on the values of the [WildcardFilter](#) tcl variable, which are case-insensitive. If you specify more than one value, you must enclose them all in quotes ("") You may optionally prepend each variable with a minus sign (-) to disable the filter for those types, allowing them to appear. You can also use the following special values in addition to the standard WildcardFilter values: UPF, Capacity, Cells.

none

Explicitly states that you want no filter applied to the results.

Some examples:

```
-bydu DU -filter cells
-bydu DU -filter "cells upf"
-bydu DU -filter "cells -capacity"
```

- **-file <file_name>**

Writes a complete list of the instances in a design to a file. Mutually exclusive with -bydu and <object_name>.

<file_name> — A string specifying the name for a file.

- <object_name> ...
Specifies the name of an instance or block to search for. Allows multiple instances and wildcard characters. Mutually exclusive with -file and -bydu.
- -protected
Filters and reduces the output to only those instances (implied by the <object_name> argument) that are protected because they meet one or both of the following requirements:
 - The instance is in a protected region.
 - The instance is in a design unit that has a protected region in it.
- -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 design unit name from any instance in the output.

Arguments for virtuals

When searching for virtuals, you must specify all optional arguments before any object names.

- <object_name> ...
(required) Specifies the virtual object to search for. Allows multiple virtuals and wildcard characters.
- -kind <kind>
(optional) Specifies the kind of virtual object to search for.

<kind> — A virtual object of one of the following kinds:
 - designs
 - explicits
 - functions
 - implicits
 - signals.
- -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 to search. Allows wildcard characters . 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 to search for. Allows wildcard characters. 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.

Examples

- Find all signals in the entire design.
find signals -r /*
- Find all instances in the entire design and save the list in the file *instancelist.txt*.
find instances -file instancelist.txt -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 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 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, you must include a space after the closing '\' and before the closing '}'.
find signals {/top/inst1/inst2/extended'}

find signals {/top\My extended id\}

- If /dut/core/pclk exists, prints the message "pclk does exist" in the transcript. This is typically 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 u1 in path  
set pattern_match "*u1*" ;  
# 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 ;
```

find connections

Returns the set of nets that are electrically equivalent to a specified net. Available only during a live simulation.

Syntax

find connections <net>

Arguments

- <net>
(required) A net in the design.

Examples

find connections /top/p/strb

returns:

```
# Connected nets for strb
#   output : /top/p/strb
#   internal : /top/pstrb
#   input : /top/c/pstrb
```

find infiles

Sets up a search for a string in the specified file(s) and prints the results to the Transcript window. The results have individual hotlinks that will open the file and display the location of the string.

Syntax

```
find infiles <string_pattern> <file>...
```

Description

When you run this command in command-line mode from outside of the GUI, the results are sent to stdout with no hotlinks.

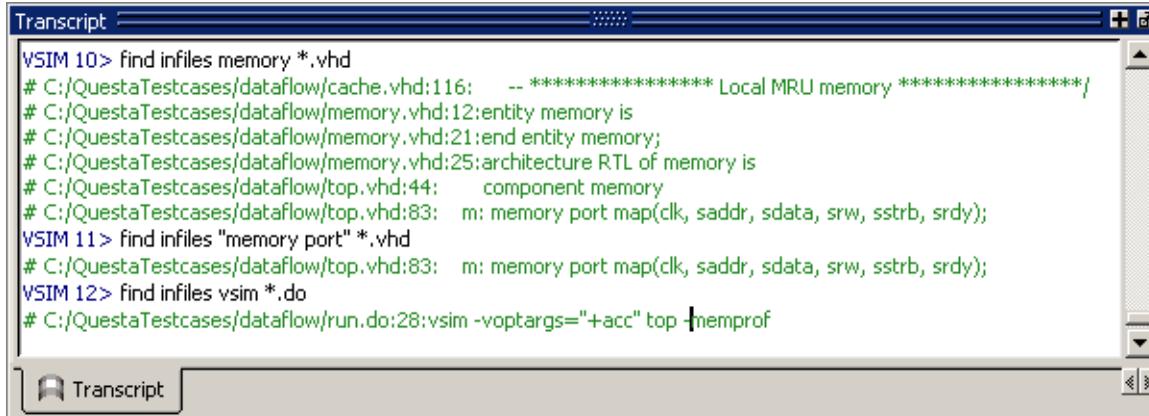
Arguments

- <string_pattern>
(required) The string to search for. Must be the first argument. You can use Tcl regular expression wildcards to further restrict the search capability.
- <file>...
(required) The file(s) to search. Must be the second argument. You can use Tcl regular expression wildcards to further restrict the search capability.

Examples

Figure 2-2 shows a screen capture containing examples and results of the find infiles command.

Figure 2-2. find infiles Example



The screenshot shows the ModelSim Transcript window with the title "Transcript". The window displays the following command-line session:

```
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

Searches for a string in the source files for the current design and prints the results to the Transcript window, with individual hotlinks to open the source and display the location of the string.

Note

 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 | -regexp] [-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 quotation marks (" ") if it includes spaces. You must specify the <pattern> at the end of the command line; any switches specified after <pattern> are not registered.

- -exact | -glob | -regexp

(optional) Defines the style of regular expression used in the <pattern>

-exact — Indicates that no characters have special meaning, 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”

-regexp — Allows Tcl regular expressions. For more information refer to the Tcl documentation:

[Help > Tcl Man Pages](#)

Select “Tcl Commands”, then “re_syntax”.

- -inline

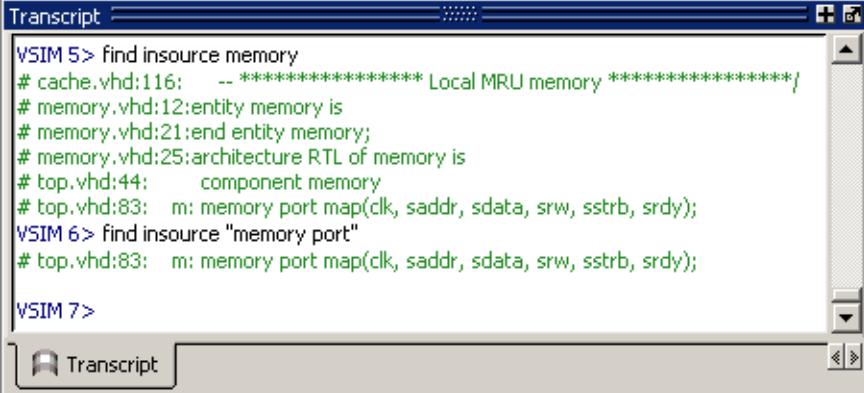
(optional) Returns the matches in the form of a Tcl list, which disables the hotspot feature, but can simplify post-processing.

- -nocase

(optional) Treats <pattern> as case-insensitive.

Examples

- [Figure 2-3](#) shows examples of the find insource command, including results.

Figure 2-3. find insource Example

The screenshot shows the ModelSim Transcript window 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>
```

- Searching for two keywords with whitespace between them:

find insource -regexp {top_dut\s+dut}

returns:

```
# top.sv:20:          top_dut          dut (
```

- Searching for string starting with 'dut' and ending with 'o':

find insource -regexp {dut.*o}

returns:

```
# top.sv:17:          DUT_io dut_io(.clock(tb_clk), .reset(tb_reset));
```

- Searching for string irrespective of case:

find insource -regexp -nocase {DUT}

returns:

```
# test.sv:10:          virtual DUT_io dut_io;
# test.sv:27:          this.dut_io = dut_io;
```

Related Topics

[DISABLE_ELAB_DEBUG \[ModelSim User's Manual\]](#)

force

Enables you to apply stimulus interactively to VHDL signals, Verilog nets and registers.

Syntax

Forcing values, driver type, repetition time or stop time on an object

```
force {<object_name> <value> [[@]<time_info>][, <value> [@]<time_info>]...  
      [-deposit | -drive | -freeze] [-cancel [@]<time_info>] [-repeat [@]<time_info>]
```

Reporting all force commands

Specifying this command without arguments 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 and Verilog.

For example, if you enter:

```
force -freeze /top/p/addr 0 100, 1 150 -r 200 -cancel 2000
```

with the specified times being relative to the current simulation time of 2820 ns,

Entering:

```
force
```

Returns:

```
# force -freeze /top/p/addr 0 {@2920 ns} , 1 {@2970 ns}  
      -repeat {@3020 ns} -cancel {@4820 ns}
```

Note

 When you run the force command, the simulator translates the relative time you specify into absolute time.

Description

You can create a complex sequence of stimuli when using the force command in a DO file.

The constraints on what you can and cannot force include:

You can force:

- VHDL signals or parts of signals.
- Verilog nets and registers, bit-selects, part-selects, and field-selects. Refer to “[Force and Release Statements in Verilog](#)” in the User’s Manual for more information.
- Virtual Signals if the number of bits corresponds to the signal value. Refer to “[Virtual Signals](#)” in the User’s Manual.
- An alias of a VHDL signal.

- An input port that is mapped at a higher level in VHDL and mixed models.

You cannot force:

- Virtual functions.
- VHDL variables. Refer to the [change](#) command for information on working with VHDL variables.
- An input port that has a conversion function on the input or on the path up the network mapped from the input.

You can use the -help switch to find additional information on this command.

Arguments

- <object_name>

(required when forcing a value change) Specifies the name of the HDL object to be forced. A wildcard is permitted only if it matches one object. Refer to [Design Object Names](#) and [Tcl Syntax and Specification of Array Bits and Slices](#) 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 can also specify a record sub-element, an indexed array, or a sliced array, as long as the type is one of the above. Must be the first argument to the force command.

- <value>

(required when forcing a value change) Specifies the value to force the object to. The specified value must be appropriate for the type. Must be the second argument to the force command.

A 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):

Description	VHDL Value	Verilog Value
character literal sequence	F	F
binary radix	2#1111	'b1111
octal radix	8#17	'o17
decimal radix	10#15	'd15
hexadecimal radix	16#F	'hF

To force a negative value, ensure that the negative sign ‘-’ precedes the base designation. For example, to force a -2 value in base 10 in VHDL, write:

```
force <object_name> -10#2
```

Note

 For based numbers in VHDL, ModelSim translates each 1 or 0 to the appropriate value for the number's enumerated type. The translation table in the *pref.tcl* file controls the translation. If ModelSim cannot find a translation for 0 or 1, it uses the left bound of the signal type (type'left) for that value.

You can create a sequence of forced values on an object by specifying <value>[@]<time_info> in a comma/space separated list.

For example:

```
force /top/p/addr 1 100ns, 0 200ns, 1 250ns
```

- -cancel [@]<time_info>

(optional) Cancels the force command at the time specified by <time_info>.

where:

<time_info> is [@]<time_value>[<time_unit>]

Refer to [\[@\]<time_info>](#) for more information about specifying time values.

- -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.

- -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. When used for registers, it behaves like the [change](#) command. Supports multi-dimensional indexing on the force target.

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. (Refer to [DefaultForceKind](#) in the User's Manual.)

- -freeze

(optional) Freezes the object at the specified <value> until it is forced again or until it is unforced with the [noforce](#) command.

Note

 If you prefer -freeze as the default for resolved and unresolved VHDL signals, change the DefaultForceKind variable in the *modelsim.ini* file. (Refer to [DefaultForceKind](#) in the User's Manual.)

- -repeat [@]<time_info>

(optional) Repeats a series of forced values and times at the time specified.

where:

<time_info> is [@]<time_value>[<time_unit>]

Refer to [\[@\]<time_info>](#) for more information about specifying time values.

You must specify at least two <value> <time_info> pairs on the forced object before specifying -repeat, for example:

```
force top/dut/p 1 0, 0 100 -repeat 200 -cancel 1000
```

A repeating force command will force a value before other non-repeating force commands that occur in the same time step.

- [@]<time_info>

(optional) Specifies the relative or absolute simulation time at which the <value> is to be applied.

where:

<time_info> is [@]<time_value>[<time_unit>]

@ — A prefix applied to <time_value> to specify an absolute time. By default, the specified time units are assumed to be relative to the current time, unless the value is preceded by the character "at" (@). Omit the "at" (@) character to specify relative time. For example:

```
-cancel {520 ns}      \\ Relative Time
```

```
-cancel {@ 520 ns}   \\ Absolute Time
```

<time_value> — 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.

<time_unit> — An optional suffix specifying a time unit where the default is to use the current simulator time by omitting <time_unit>. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr.

<time_value> and <time_unit> can be formatted in any of the following ways:

```
10ns
10 ns
{10 ns}
"10 ns"
```

Note

 If you specify a sequence of forces and use braces ({}) surrounding a <time_value> and <time_unit> pair, you must place a space in front of the comma (,) separating the two value/time pairs. For example:

```
force foo 1 {10 ns} , 0 {20 ns}
```

Examples

- Reporting all recently applied force commands

If you specify this command with no arguments, it returns a list of all forced objects and the changes applied. For example, after executing:

```
force -freeze /top/p/addr 0 100, 1 150 -r 200 -cancel 2000
```

where the times specified are relative to the current simulation time, in this case 2820 ns.

Entering:

```
force
```

returns:

```
# force -freeze /top/p/addr 0 {@2920 ns} , 1 {@2970 ns}
  -repeat {@3020 ns} -cancel {@4820 ns}
```

Note

 Executing the force command translates the relative time you specified into absolute time.

- Force input1 to 0 at the current simulator time.

```
force input1 0
```

- Force the fourth element of the array bus1 to 1 at the current simulator time.

```
force bus1(4) 1
```

- Force bus1 to 01XZ at 100 nanoseconds after the current simulator time.

```
force bus1 2#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. Repeat this cycle every 100 time units after the current simulation time. If the current simulation time is 100 ns, the next transition is to 1 at 110 ns and 0 at 120 ns, this pattern to start repeating at 200 ns.

```
force input1 1 10, 0 20 -r 100
```

- Similar to the previous example, but also specifies the time units.

```
force input1 1 10 ns, 0 20 ns -r 100 ns
```

- Force signal s to alternate between values 1 and 0 every 100 time units until 1000 time units have occurred, starting from time Now. 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
}
```

- Force one bit of a record containing an array.

```
force struct1.bus1(4) 1
```

- Force a slice of an array.

```
force {bus1[2:5]} 'hF
```

Related Topics

[DefaultForceKind \[ModelSim User's Manual\]](#)

[Force and Release Statements in Verilog \[ModelSim User's Manual\]](#)

[Force Command Defaults \[ModelSim User's Manual\]](#)

[noforce](#)

[Virtual Signals \[ModelSim User's Manual\]](#)

formatTime

Provides global format control for all time values displayed in the GUI. When specified without arguments, 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 displays 6,458,131 ps.

- Use largest unit value possible.

formatTime +bestunits

Displays 8 us instead of 8,000 ns.

gc configure

Prerequisite:

Before using this command, do one of the following:

- Regular simulation — Simulate with the vsim command, but omit the -classdebug argument.
- Interactive class debugging — Simulate with the vsim -classdebug command.

Specifies when to run the System Verilog garbage collector. Returns the current settings when specified without arguments.

Syntax

```
gc config [-onrun 0 | 1] [-onstep 0 | 1] [-threshold <n>]
```

Arguments

- **-onrun 0 | 1**
(optional) Enables or disables post-simulation run garbage collection.
 - 0 — Off, default for regular simulation
 - 1 — On, default for interactive class debugging
- **-onstep 0 | 1**
(optional) Enables or disables post-step garbage collection.
 - 0 — Off, default for both regular simulation and interactive class debugging.
 - 1 — On
- **-threshold <n>**
(optional) Sets the maximum amount of memory in megabytes allocated for storage of class objects that, when exceeded, causes the garbage collector to run.
 - <n> — Any positive integer where <n> is the number of megabytes.
 - Regular simulation default = 100 megabytes
 - Interactive class debugging default = 5 megabytes

Description

You can configure the garbage collector to run after a memory threshold has been reached, after each simulation run command completes, and/or after each simulation step command. The default settings are optimized to balance performance and memory usage for either regular simulation or class debugging (vsim -classdebug).

Related Topics

[SystemVerilog Class Debugging \[ModelSim User's Manual\]](#)

[Class Instance Garbage Collection \[ModelSim User's Manual\]](#)

[GCThreshold \[ModelSim User's Manual\]](#)

[GCThresholdClassDebug \[ModelSim User's Manual\]](#)

gc run

Runs the SystemVerilog garbage collector.

Syntax

`gc run`

Arguments

None

Related Topics

[gc configure](#)

[SystemVerilog Class Debugging \[ModelSim User's Manual\]](#)

[Class Instance Garbage Collection \[ModelSim User's Manual\]](#)

[GCThreshold \[ModelSim User's Manual\]](#)

[GCThresholdClassDebug \[ModelSim User's Manual\]](#)

help

Displays a brief description and syntax for the specified command in the Transcript window.

Syntax

help [[<command>](#) | [<topic>](#)]

Arguments

- <command>
(optional) Specifies the command to provide help for. The entry is case and space sensitive.
- <topic>
(optional) Specifies a topic to provide help for. 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

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.

layout

Enables you to perform a number of editing operations on custom GUI layouts, such as loading, saving, maximizing, and deleting.

Syntax

```
layout active
layout current
layout delete <name>
layout load <name>
layout names
layout normal
layout maximized
layout restoretpe <window>
layout save <name>
layout showsuppresstypes
layout suppresstype <window>
layout togglezoom
layout zoomactive
layout zoomwindow <window>
```

Description

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 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 restoretpe — removes the list of window type(s) that will not be restored when a new layout is loaded.
- layout save – saves the current layout to the specified name

- layout showsuppresstypes — lists the window types that will not be restored when a new layout is loaded.
- layout suppresstype — adds the specified window type(s) to the list of types that will not be restored when a layout is reloaded.
- 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

Arguments

- <name>
(required) Specifies the name of the layout.
- <window>
(required) The window specification can be any format accepted by the [view](#) command. You can specify the window by its type (such as wave, list, objects), by the windowobj name (such as main_pane.wave, .main_pain.library), or by the tab name (such as wave1, list3)

Related Topics

[Simulator GUI Layout Customization \[ModelSim GUI Reference Manual\]](#)

[Configuring Default Windows for Restored Layouts \[ModelSim GUI Reference Manual\]](#)

log

Creates a wave log format (WLF) file containing simulation data for all HDL objects whose names match the provided specifications.

Syntax

```
log [-howmany] [-filter <f> | -nofilter <f>] {[-in] [-inout] [-out] | [-ports]} [-internal] [-recursive]
```

```
[-depth <level>]] <object_name> ...
```

```
log -flush [<object>]
```

Description

Note

 The log command is also known as the “add log” command.

By default the file is named vsim.wlf and stored in the current working directory. You can change the default name using the vsim -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 automatically records 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.

Arguments

- **-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.

- **-filter <f> | -nofilter <f>**

(optional) Allows a one-time modification of the WildcardFilter in the command invocation. The add log command can take as many [-filter <f>] and [-nofilter <f>] arguments as you would like to specify. Valid filters, <f>, exactly match the set of words

that can be applied to the WildcardFilter. The command uses the WildcardFilter first, then applies any user specified filters. The -filter values are added to the filter, the -nofilter values are removed from the filter. Filters are applied in the order specified, so any conflicts are resolved with the last specified wins.

- **-flush [<object>]**
(optional) Forces the WLF file to write all buffered region and event data to the WLF file. By default, the region and event data is buffered and periodically written to the file, as appropriate. Specifying <object> logs that object, and then flushes the file.
- **-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.
- **-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. Omitting -recursive limits the search 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 to log. Must be the final argument to the log command. Specify multiple object names as a space-separated list. Allows wildcard characters. 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 | +nolibcell 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

[dataset alias](#)

[dataset clear](#)

[dataset close](#)

[dataset config](#)

[dataset info](#)

[dataset list](#)

[dataset open](#)

[dataset restart](#)

[dataset rename](#)

[dataset save](#)

[dataset snapshot](#)

[nolog](#)

[Recording Simulation Results With Datasets \[ModelSim User's Manual\]](#)

[WLFFilename \[ModelSim User's Manual\]](#)

lshift

Takes a Tcl list as an argument, and shifts it one place to the left, eliminating the left-most element.

Syntax

`lshift <list> [<amount>]`

Description

You can specify the number of shift places to be a number greater than one. Returns nothing.

Arguments

- `<list>`
(required) Specifies the Tcl list to target with lshift. Must be the first argument to the lshift command.
- `<amount>`
(optional) Specifies the number of places to shift. The default is 1.

Examples

```
proc myfunc args {  
    # throws away the first two arguments  
    lshift args 2  
    ...  
}
```

I sublist

Returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern.

Syntax

I sublist <list> <pattern>

Arguments

- <list>
(required) Specifies the Tcl list to target with I sublist. Must be the first argument.
- <pattern>
(required) Specifies the pattern to match within the <list> using Tcl glob-style matching.
Must be the final argument.

Examples

- In the example below, variable 't' returns "structure signals source".

```
set window_names "structure signals variables process source wave list"  
set t [I sublist $window_names s*]
```

mem compare

Compares a selected memory to a reference memory or file.

Note

 The "diff" utility must be installed and be visible in your search path in order to run this command.

Syntax

```
mem compare {[ -mem <ref_mem> ] | [ -file <ref_file> ]} [actual_mem]
```

Arguments

- **-mem <ref_mem>**
(optional) Specifies a reference memory to compare with actual_mem.
 <ref_mem> — A memory record.
- **-file <ref_file>**
(optional) Specifies a reference file to compare with actual_mem.
 <ref_file> — A saved memory file.
- **actual_mem**
(required) Specifies the name of the memory to compare against the reference data. Must be the final argument to the mem compare command.

mem display

Prints the memory contents of the specified instance to the Transcript window.

Note

 You do not need to specify the signal or variable name if the given instance path contains only a single array signal or variable.

Syntax

```
mem display [-addressradix [d | h]] [-compress] [-dataradix <radix_type>]  
[-endaddress <end>][-format [bin | hex | mti]] [-noaddress] [-startaddress <st>]  
[-wordsperline <n>] [<path>]
```

Description

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 you specify Verilog memory format (hex or binary).

You can specify the address radix, data radix, and address range for the output, 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

- **-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 to not print identical lines. 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 you do not specify a radix for an enumerated type, the symbolic representation is used.

You can use the [radix](#) command to change the default radix type for the current simulation, or edit the DefaultRadix variable in the modelsim.ini file to make the default radix permanent. Refer to [DefaultRadix](#) in the User's Manual for more information.

- **-endaddress <end>**
(optional) Specifies the end address for a range of addresses to display.
 <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 not to print addresses.
- **-startaddress <st>**
(optional) Specifies the start address for a range of addresses to display.
 <st> — Any valid address in the memory. If unspecified, the default is the start of the memory.
- **-wordsperline <n>**
(optional) Specifies how many words to print 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. Must be the final argument to the mem display command. The default is the current context, as shown in the Structure window. You can specify indexes.

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
```

mem display

- 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

[mem load](#)

mem list

Displays a flattened list of all memory instances in the current or specified context after a design has been elaborated.

Syntax

mem list [-r] [<path>]

Description

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.

Arguments

- **-r**
(optional) Specifies to recursively descend into sub-modules when listing memories.
- **<path>**
(optional) Specifies 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

Updates the simulation memory contents of a specified instance.

Syntax

```
mem load {-infile <infile> | -filldata <data_word> [-infile <infile>]} [-endaddress <end>]
[-fillradix <radix_type>] [-filltype {dec | inc | rand | value}] [-format [bin | hex | mti]]
[<path>] [-skip <Nwords>] [-startaddress <st>] [-truncate]
```

Description

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 you do not specify an address range (starting and ending address), the memory is loaded starting at the first location.

You can upload contents either from a memory data file, a memory pattern, or both. If you specify both, the pattern is applied only to memory locations not contained in the file.

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 to 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 depends on the -filltype specified:

- For fixed pattern values, the fill pattern is repeatedly tiled to initialize the specified memory block. 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 is 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. You can specify a seed value on the command line. For any given seed, the generated sequence is identical.

The pattern data is interpreted according to the default system radix setting. However, you can override this setting with a standard Verilog-style '<radix_char><data>' specification.

Arguments

- **-infile <infile>**
(required unless the -filldata 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 load.
<end> — Specified as any valid address in the memory.
- **-filltype {dec | inc | rand | value}**
(optional, use with the -filldata argument) Fills in memory addresses in an algorithmic pattern, starting with the data word specified in -filldata. Specifying a fill pattern without a file option fills the entire memory or specified address range with the specified pattern. If you specify both, the pattern applies only to memory locations not contained in the file.
 - 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 unless -infile is used) Specifies the data word to use to fill memory addresses in the pattern specified by -filltype.
<data_word> — Specifies the data word. Must be in the same format specified by the -fillradix switch.
- **-fillradix <radix_type>**
(optional, use with -filldata) Specifies the 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, use with -infile) Specifies the format of the file to load.
 - bin* — Specifies binary data format.
 - hex* — Specifies hex 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.

The MTI memory data file format stores internal file address and data radix settings within the file itself, so you do not 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, you can use shorthand instance names. The default is the current context, as shown in the Structure window.
You can specify memory address indexes 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 skip 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 load.
 <st> — Any valid address in the memory.
- -truncate
(optional) Ignores any most significant bits (msb) in a memory word that 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: 000000000000100 000000000000101 000000000000110
000000000000111
# 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
```

- Load memory instance /top/mem with zeros (0).

```
mem load -filldata 0 /top/mem
```

- 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

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.

Syntax

```
mem save -outfile <filename> [-addressradix {dec | hex}] [-dataradix <radix_type>]  
[-format {bin | hex | mti}] [-compress | -noaddress] [<path>]  
[-startaddress <st> -endaddress <end>] [-wordsperline <Nwords>]
```

Description

This command works identically to the mem display command, except that it writes output to a file rather than a display.

The -format argument specifies the order in which data is written to the file. 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.

Arguments

- **-outfile <filename>**
(required) Specifies to store the memory contents in a file.
 <filename> — The name of the file in which to store the specified memory contents.
- **-addressradix {dec | hex}**
(optional) Specifies the address radix for the default mti formatted files.
 dec — Decimal (default).
 hex — Hexadecimal.
- **-compress**
(optional) Specifies to print only unique lines, and not identical lines. 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 use the [radix](#) command to change the default radix for the current simulation. You can edit the DefaultRadix variable in the modelsim.ini file to change the default radix permanently . Refer to [DefaultRadix](#) in the User's Manual for more information.

- **-endaddress <end>**

(optional) Specifies the end address for a range of addresses to save.

 <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. You can use these formats with Verilog memories, and with VHDL memories composed of std_logic types.

The MTI memory data file format stores internal file address and data radix settings 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 save.

 <st> — Any valid address in the memory.

- **-wordsperline <Nwords>**

(optional) Specifies how many memory values to print 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 0000000000000001 0000000000000010
0000000000000011
4: 000000000000100 000000000000101 000000000000110
000000000000111
8: 0000000000001000 0000000000001001 xxxxxxxxxxxxxxxxx
```

Related Topics

[mem display](#)

[mem load](#)

mem search

Finds and prints to the screen the first occurring match, or all matches, of a specified memory pattern in the specified memory instance.

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. Specify multiple word patterns as a space separated list.
Accepts wildcards.

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. Accepts wildcards. Specify multiple word patterns 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. The default is to print only the first matching occurrence.

- **-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 use the [radix](#) command to change the default radix for the current simulation. You can change the default radix permanently by editing the DefaultRadix variable in the modelsim.ini file. Refer to [DefaultRadix](#) in the User's Manual for more information.

- **-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. Specify multiple word patterns as a space-separated list. 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 -addressradix hex -all /uut/u1/mem3
```

returns:

```
#1ea5: 2750
#1ea6: 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>

modelsim

Starts the ModelSim GUI without prompting you to load a design.

Syntax

`modelsim [-do <macrofile>] [<license_option>] [-nosplash]`

Description

This command is valid only for Windows platforms, and can be invoked:

- 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. (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`.

Arguments

- `-do <macrofile>`
(optional) Executes a DO file when `modelsim` is invoked.
`<macrofile>` — The name of a DO file.

Note

 In addition to the script called by this argument, invoking `vsim` calls any DO file specified by the STARTUP variable in *modelsim.ini*.

- `<license_option>`
(optional) Restricts the search of the license manager.
- `-nosplash`
(optional) Disables the splash screen.

Related Topics

[do](#)

[Using a Startup File \[ModelSim User's Manual\]](#)

noforce

Removes the effect of any active force commands on the selected HDL objects, and also causes the object's value to be re-evaluated.

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. You can specify multiple object names as a space-separated list. Allows wildcard characters.

Related Topics

[force](#)

nolog

Suspends writing of data to the wave log format (WLF) file for the specified signals.

Syntax

```
nolog [-all] [-depth <level>] [-howmany] [-in] [-inout] [-internal] [-out] [-ports] [-recursive]  
[-reset] [<object_name>...]
```

Description

Writes a flag 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. You can turn logging back on by issuing another [log](#) command or by doing a nolog -reset.

Because the nolog command adds new information to the WLF file, older versions of the simulator cannot read WLF files created when using the nolog command.

Transactions written in SCV or Verilog are logged automatically, and you can remove them with the nolog command. A transaction is logged into the .wlf file if logging is enabled for that stream, and hasn't been disabled with nolog, when the transaction begins. The entire span of a transaction is either logged or not logged, regardless of the begin and end times specified for that transaction.

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. Omitting -recursive limits the search 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 to unlog. You can specify multiple object names as a space-separated list. Allows wildcard characters.

Examples

- Unlog all objects in the design.
nolog -r /*
- Turn logging back on for all unlogged signals.
nolog -reset

Related Topics

[log](#)

notepad

Opens a simple text editor to view and edit ASCII files or create new files.

Syntax

`notepad [<filename>] [-r | -edit]`

Description

You can change this mode from the Notepad Edit menu.

Returns nothing.

Arguments

- `<filename>`
(optional) Name of the file to display.
- `-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

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. You can specify multiple window types in a space-separated list. Allows wildcards. Requires at least one type (or wildcard).

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

nowhen

Deactivates selected when commands.

Syntax

nowhen [[<label>](#)]

Arguments

- [<label>](#)
(optional) Specifies an individual when command. You can use wildcards to select more than one when command.

Examples

- Deactivate the [when](#) command labeled 99.

nowhen 99

- Deactivate all [when](#) commands.

nowhen *

onbreak

Used in a DO file to specify one or more scripts to execute when the simulation encounters a breakpoint in the source code.

Syntax

```
onbreak <script>; <script>...
```

Description

You must include a [run](#) or [step](#) command after any `onbreak` command in your DO file.

An `onbreak` script affects any subsequent run commands, until another `onbreak` command is issued. If a script executes a DO file, the contents of the DO file inherit the calling script's `onbreak` setting, unless and until the DO file executes another `onbreak` command. In that case, the new `onbreak` setting takes effect until the DO file script completes. Execution then returns to the calling script, restoring the calling script's `onbreak` setting. (Refer to [Breakpoint Flow Control in Nested DO files](#) in the User's Manual for more information.)

The `OnBreakDefaultAction` Tcl variable defines the default behavior when there is no `onbreak` setting in effect. If the `OnBreakDefaultAction` variable is not defined, the simulator default is to resume execution.

You can use an empty string to change the `onbreak` command back to the default behavior:

```
onbreak ""
```

In this case, the script resumes after a breakpoint occurs (after any associated [bp](#) command string is executed).

If you do not specify an argument to the `onbreak` command, it returns either the definition of any scripts defined by previous `onbreak` commands, or nothing, if you have not previously defined any `onbreak` commands.

Arguments

- `<script>`

(optional) Any command or script can be used as an argument to `onbreak`. To use more than one command or script, separate them by semicolons, or place them on multiple lines and enclose the entire script in braces (`{ }`) or quotation marks (`" "`). If a `run` or `step` command is issued within an `onbreak` script, the script returns immediately, and without executing any of the following commands. It is an error to execute any commands following a `run` command within an `onbreak` command string. 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 DO 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](#) in the User's Manual). 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 {
    lassign [runStatus -full] status fullstat
    if {$status eq "error"} {
        # Unexpected error, report info and force an error exit
        echo "Error: $fullstat"
        set broken 1
        resume
    } elseif {$status eq "break"} {
        # If this is a user break, then
        # issue a prompt to give interactive
        # control to the user
        if {[string match "user_*" $fullstat]} {
            pause
        } else {
            # Assertion or other break condition
            set broken 2
            resume
        }
    } else {
        resume
    }
}
run -all
if {$broken == 1} {
    # Unexpected condition. Exit with bad status.
    echo "failure"
    quit -force -code 3
} elseif {$broken == 2} {
    # Assertion or other break condition
    echo "error"
    quit -force -code 1
} else {
    echo "success!"
}
quit -force
```

Related Topics

[do](#)

[onerror](#)

[Useful Commands for Handling Breakpoints and Errors \[ModelSim User's Manual\]](#)

[DO Files \[ModelSim User's Manual\]](#)

onElabError

Used in a DO file script, specifies one or more commands to execute when an error is encountered during the elaboration portion of a vsim command.

Syntax

```
onElabError {[<command> [<command>] ...]}
```

Arguments

- <command>

(optional) Any command can be used as an argument. If you use more than one command, separate them with semicolons, or place them on multiple lines. You must enclose the entire command string in braces ({}). If you do not enter any arguments, onElabError returns to a prompt.

Related Topics

[do](#)

onerror

Used in a DO file before a run command, specifies one or more commands to execute when a running script encounters an error.

Syntax

```
onerror {[<command> [; <command>] ...]}
```

Description

Using the onerror command without arguments returns 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 DO file.

With a successful onerror command, the DO file script continues normally, unless the command instructs the simulator to quit, as for example:

```
onerror{quit -f}
```

or

```
onerror {break}
```

An unsuccessful onerror command causes the simulator to quit. For example:

```
onerror {add wave b}
```

when you do not have a signal named b.

The onerror command executes when a Tcl command (such as break) encounters an error in the DO file that contains the onerror command (note that a run command does not necessarily need to be in process).

Arguments

- <command>
(optional) Specifies the command or commands to run. Separate multiple commands with semicolons, or place them on multiple lines. You must enclose the entire command string in braces ({}).

Examples

- Force the simulator to quit if an error is encountered while the DO file is running.

```
onerror {quit -f}
```

Related Topics

[do](#)

[onbreak](#)

[Useful Commands for Handling Breakpoints and Errors \[ModelSim User's Manual\]](#)

[DO Files \[ModelSim User's Manual\]](#)

onfinish

Controls simulator behavior when encountering \$finish or sc_stop() in the design code.

Note

 Specifying this command without an argument returns the current setting.

Syntax

`onfinish [ask | exit | final | stop | default]`

Description

The onfinish command controls how the simulation behaves when encountering a \$finish or sc_stop() statement. It can either function as a stop command or exit the simulator. Stopping can aid post-simulation debug by allowing you to examine the state of the design prior to exiting.

Arguments

- ask
 - (optional) In batch mode, the simulation will exit; in GUI mode, the user is prompted for action to either stop or exit the simulation.
- exit
 - (optional) The simulation exits without asking for any confirmation.
- final
 - (optional) The simulation executes the stop command *after* executing any SystemVerilog final blocks.
- stop
 - (optional) The simulation executes the stop command *before* executing any SystemVerilog final blocks.
- default
 - (optional) Uses the current setting for the OnFinish variable in the *modelsim.ini* file.

Related Topics

[OnFinish \[ModelSim User's Manual\]](#)

pause

Inserted in a macro, interrupts execution of the macro, allowing you to perform interactive debugging.

Syntax

`pause`

Description

When a macro is interrupted during execution, the macro returns the prompt:

`VSIM(paused) >`

to notify 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 invoke a third macro, and you can continue invoking macros up to a nesting level of 50 macros.

Use the [status](#) command to list summary information about all interrupted macros.

Use the [resume](#) command to resume execution of a macro.

Use the [abort](#) command to stop execution of some or all of the macros.

Arguments

None.

Related Topics

[resume](#)

[run](#)

[status](#)

precision

Determines how real numbers display in the graphic interface (such as the Objects, Wave, Locals, and List windows).

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.

Description

The precision command does not affect the internal representation of a real number and therefore does not allow precision values over 17. Executing the command without any arguments returns the current precision setting.

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 does not print trailing zeros.

precision 8

For example:

1.23456 or 6543.21

printenv

Prints the current names and values of all environment variables to the Transcript window. 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

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 in which to save raw process data. If you do not specify -file, 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 you do not use this option, the process data overwrites the existing process report file.

project

Used to perform common operations on projects.

Syntax

```
project [addfile <filename> [<file_type>] [<folder_name>]] | [addfolder <foldername> [<folder_parent>]] | [calculateorder] | [close] | [compileall [-n]] | [compileorder] | [compileoutofdate [-n]] | [delete <filename>] | [filenames] | [env] | [history] | [new <home_dir> <proj_name> [<defaultlibrary>] [<intialini>] [0 | 1]] | [open <project>] | [removefile <filename>]
```

Description

Some arguments to this command are applicable only if you have opened a project with either the project new or project open command. Some arguments are applicable only outside of a simulation session. Refer to the argument descriptions for more information.

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 you do not specify a folder name, the file is 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 process repeats 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.

- **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.

- **filenames**

Returns the absolute pathnames of all files contained in the currently open project.

- **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>] [<intialini>] [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 defaults to "work" unless otherwise specified. You cannot create a new project while a project is currently open, or while 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 is saved as an .mpf file

<defaultlibrary> — Specifies a name for the default library.

<intialini> — Specifies an optional *modelsim.ini* file as a seed for the project file. If initialini is an empty string, 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 the 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 change 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

pwd

A Tcl command; displays the current directory path in the Transcript window.

Syntax

`pwd`

Arguments

None

quietly

Turns off transcript echoing for the specified command.

Syntax

`quietly <command>`

Arguments

- `<command>`

(required) Specifies the command to disable transcript echoing for. Results that are normally echoed will no longer appear in the Transcript window. To disable echoing for all commands use the [transcript](#) command with the -quietly option.

Related Topics

[transcript](#)

quit

Exits the simulator.

Note

 Do not use either the quit command or an [exit](#) command to stop the simulation using a [when](#) command. Instead, you must use a [stop](#) command in your when statement. 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. Closes all files opened by the simulation, including the WLF file (*vsim.wlf*).
- **-code <integer>**
(optional) Quits the simulation and issues an exit code.

 <integer> — The value of the exit code. Do not specify an exit code that already exists in ModelSim. Refer to "[Exit Codes](#)" in the User's Manual for a list of existing exit codes. You can also specify a variable in place of <integer>.

Best practice is to 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 command -code arguments behave the same way.

radix

Specifies the default radix to use for the current simulation.

Syntax

```
radix [-binary | -octal | -decimal | -hexadecimal | -HEXADECIMAL | -unsigned | -ascii | -time]
      [-enumnumeric | -enumsymbolic | -showbase | -symbolic | -wreal]
```

Description

You can use the radix command at any time. Specify radix with no argument to return the current radix setting.

The specified radix is used for all commands (such as [force](#), [examine](#), [change](#)), and for values displayed in the Objects, Locals, Dataflow, List, and Wave windows, as well as the Source window in source annotation view.

There are two alternate methods for changing the default radix:

- Edit the DefaultRadix variable in the *modelsim.ini* file. (Refer to [DefaultRadix](#) in the User's Manual for more information.)
- Choose **Simulate > Runtime Options** from the main menu, click the **Defaults** tab, make your selection in the **Default Radix** box.

Numeric radix, other than symbolic, translate bits to a 4-state representation as part of the numeric conversion. Groups of bits are then converted to a number in the correct radix, or to 'x' or 'z' if the value is not numeric (that is, contains only '0's and '1's). There is no 'U' in the 4-state representation, nor 'W' or '-'. All odd values are converted to 'x'.

Alternatives to changing the default radix for the simulation session include:

- Use [examine](#) <name> -<radix_type> to transcript the current value of <name> using the specified radix.
- Use [radix signal](#) to set a signal-specific radix.

Arguments

You can abbreviate the following arguments to any length. For example, -dec is equivalent to -decimal.

- -ascii
 - (optional) Display a Verilog object as a string equivalent using 8-bit character encoding.
- -binary
 - (optional) Displays values in binary format.

- **-enumnumeric**
(optional) Causes Verilog enums to display 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 enums as symbols. Reverses the action of the -enumnumeric option.
- **-decimal**
(optional) Displays values in decimal format. You can specify -signed as an alias for this argument.
- **-hexadecimal**
(optional) Displays values in hexadecimal format using lower-case a-f characters.
- **-HEXADECIMAL**
(optional) Case-sensitive argument that displays values using upper-case A-F characters.
- **-octal**
(optional) Displays values in octal format.
- **-time**
(optional) Displays values of time for register-based types in Verilog.
- **-showbase**
(optional) Displays the number of bits of the vector and the radix used (binary = b, decimal = d, hexadecimal = h, ASCII = a, and time = t)
For example, a vector value of “31” might display as “16’h31” to show that the vector is 16 bits wide, with a hexadecimal radix.
- **-symbolic**
(optional) Displays values in a form closest to their natural form.
- **-unsigned**
(optional) Displays values in unsigned decimal format.
- **-wreal**
(optional) Displays internal values of real numbers that are determined to be not-a-number (NaN) as X or Z instead of as "nan." Internal values of NaN that match `wrealZState, display a 'Z'; otherwise, an 'X' is displayed.

Related Topics

[User-Defined Radices \[ModelSim GUI Reference Manual\]](#)

[radix define](#)

[radix delete](#)

[radix names](#)

[radix list](#)

[radix signal](#)

radix define

Creates or modifies a user-defined radix.

Syntax

User Custom Radix

```
radix define <name> <definition_body> [-color <value>]
```

Fixed or Floating Point Number Radix

```
radix define <name> [[-fixed [-signed] | -float] -fraction <n>] [-base <base>] [-precision <p>]
```

Arguments

- <name>
(required) User-specified name for the radix.
- <definition_body>
(required for custom radix) A list of number pattern, label pairs. The definition body has the form:

```
{  
    <numeric-value> <enum-label> [-color <color>],  
    <numeric-value> <enum-label>  
    -default <radix_type>  
    -defaultcolor <color>  
}
```

- <numeric-value> is any legitimate HDL integer numeric literal. To be more specific:

```
<integer>  
<base>#<value>#      --- <base> is 2, 8, 10, or 16  
<base>"value"        --- <base> is B, O, or X  
<size>'<base><value>  --- <size> is an integer,  
                           <base> is b, d, o, or h.
```

You can use the question mark (?) wildcard character for bits or characters of the value. For example:

```
radix define bus-state {  
    6'b01??00 "Write" -color orange,  
    6'b10??00 "Read" -color green  
}
```

In this example, the first pattern matches "010000", "010100", "011000", and "011100". In case of overlaps, the first matching pattern is used, going from top to bottom.

- <enum-label> Any arbitrary string. Enclose it in quotation marks (""), especially if it contains spaces.
- The comma (,) in the definition body is optional.

- -color <color> Sets the color to use to display the specific value in the Wave window.
- -default <radix_type> (optional) 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.
- -defaultcolor <color> (optional) Defines the default color to use if none has been defined for a specific value.

Refer to the Verilog and VHDL Language Reference Manuals for exact definitions of these numeric literals.

- -base <base>
(optional for fixed and floating point radices) Specifies the base for a fixed or floating point radix.

 <base> — Any valid radix type: binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default.
- -color <value>
(optional for custom radices) Designates a color for the waveform and text in the Wave window.

 <value> — The color value can be a color name or its hex value (see example in the Example section, below).
- -fixed
(required for fixed point radix) Specifies a fixed number radix.
- -float
(required for floating point radix) Specifies a floating point number radix.
- -fraction <n>
(required for fixed and floating point radices) Specifies the location of the decimal point in a vector.

 <n> — Any integer between 3 and the full bit value of the vector. For example, specifying -fraction 3 for the eight bit vector “10001001” places the decimal two bits away from the least significant bit on the right, so the vector becomes “10001.001”.
- -precision <p>
(optional for fixed and floating point radices) Specifies the number of places after the decimal point or significant digits of a floating point or fixed number in symbolic format.

 <p> — A number, less than or equal to 17, and an optional format specification, taking the form “<width>[efg],” for example 3g, 4f, or 6.
- -signed
(optional for fixed point radix) Treats fixed numbers as signed, where the most significant bit is the sign bit. The default is an unsigned number.

Description

You can use a user definable radix to map bit patterns to a set of enumeration labels or to set up a fixed or floating point radix. User-defined radices are available for use in most windows, and with the examine command.

Examples

- 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'b000000000001 "IDLE",  
    11'b000000000010 "CTRL",  
    11'b000000000100 "WT_WD_1",  
    11'b000000001000 "WT_WD_2",  
    11'b000000010000 "WT_BLK_1",  
    11'b000000100000 "WT_BLK_2",  
    11'b000010000000 "WT_BLK_3",  
    11'b000100000000 "WT_BLK_4",  
    11'b001000000000 "WT_BLK_5",  
    11'b010000000000 "RD_WD_1",  
    11'b100000000000 "RD_WD_2",  
    11'bzzzzzzzzzzzz "UNCONNECTED",  
    11'bxxxxxxxxxxx "ERROR",  
    -default hex  
}
```

Note

 The ‘z’ and ‘x’ values must be lower case.

- Specify the radix color:

```
radix define States {  
    11'b000000000001 "IDLE" -color yellow,  
    11'b000000000010 "CTRL" -color #ffee00,  
    11'b000000000100 "WT_WD_1" -color orange,  
    11'b000000001000 "WT_WD_2" -color orange,  
    11'b000000010000 "WT_BLK_1",  
    11'b000001000000 "WT_BLK_2",  
    11'b000010000000 "WT_BLK_3",  
    11'b000100000000 "WT_BLK_4",  
    11'b001000000000 "WT_BLK_5",  
    11'b010000000000 "RD_WD_1" -color green,  
    11'b100000000000 "RD_WD_2" -color green,  
    -default hex  
    -defaultcolor white  
}
```

If a pattern/label pair does not specify a color, the normal wave window colors are used. If the value of the waveform does not match any pattern, the -default radix and -defaultcolor are used.

- Create a fixed point radix named fx5 and apply that radix to the signal checksf.

Entering

VSIM> radix define fx5 -fixed -fraction 3 -base decimal -signed

returns

#fx5

Entering

VSIM> radix signal checksf

returns

#fx5

Entering

VSIM> examine -name checksf

returns

#/test_fixed/basictest/checksf -15.8750

Related Topics

[User-Defined Radices \[ModelSim GUI Reference Manual\]](#)

[precision](#)

[radix](#)

[radix delete](#)

[radix names](#)

[radix list](#)

[radix signal](#)

radix delete

Removes 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 \[ModelSim GUI Reference Manual\]](#)

[radix](#)

[radix define](#)

[radix list](#)

[radix names](#)

[radix signal](#)

radix list

Returns the complete definition of a radix, if a name is given. Lists all defined radices, if no name is given.

Syntax

`radix list [<name>]`

Arguments

- `<name>`
(optional) Returns the complete definition of the named radix.

Related Topics

[User-Defined Radices \[ModelSim GUI Reference Manual\]](#)

[radix](#)

[radix define](#)

[radix delete](#)

[radix names](#)

[radix signal](#)

radix names

Returns a list of currently defined radix names.

Syntax

`radix names`

Arguments

None

Related Topics

[User-Defined Radices \[ModelSim GUI Reference Manual\]](#)

[radix](#)

[radix define](#)

[radix delete](#)

[radix list](#)

[radix signal](#)

radix signal

Sets or inspects radix values for the specified signal in the Objects, Locals, Schematic, and Wave windows, or returns a list of all signals with a radix, when entered with no argument.

Note

 This command is most useful for a small number of signals. If the majority of signals in a design are to use a particular radix value, use the [radix](#) command to set that value as the default radix, and use the radix signal command for the rest.

Syntax

```
radix signal [<signal_name> [<radix_value>]] [-showbase]
```

Arguments

- <signal_name>
(optional) Name of the signal for which to set the radix (if <radix_value> is specified) or inspected.
- <radix_value>
(optional) Value of the radix to set for the specified signal. Use empty quotation marks ("") to unset the radix for the specified signal.
- -showbase
(optional) Display the number of bits of the vector, and the radix used (binary = b, decimal = d, hexadecimal = h, ASCII = a, and time = t).

For example, instead of displaying a vector value of “31”, display a value of “16'h31” to show that the vector is 16 bits wide, with a hexadecimal radix.

Related Topics

[User-Defined Radices \[ModelSim GUI Reference Manual\]](#)

[radix](#)

[radix define](#)

[radix list](#)

[radix delete](#)

readers

Displays the names of all readers of the specified object.

Syntax

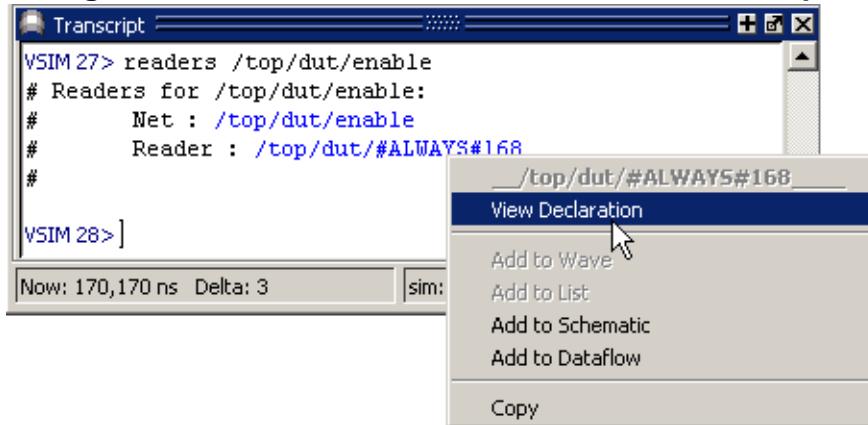
readers <object_name> [-source]

Description

The reader list is expressed relative to the top-most design signal/net connected to the specified object.

The output from the readers command displays in the Transcript window as hypertext links. You can right-click a link to open a drop-down menu and add signals to various windows. The drop-down menu includes a "View Declaration" item to open the source definition of the signal.

Figure 2-4. readers Command Results in Transcript



Arguments

- <object_name>
(required) Specifies the name of the signal or net to show readers for. All signal or net types are valid. Accepts multiple names and wildcards.
- -source
(optional) Returns the source file name and line number for each driver of the specified signal or net. If the source location cannot be determined, the value n/a is returned for that driver.

report

Displays information relevant to the current simulation.

Note

 The report command does not display information on preference variables. To view preference variables, select **Tools > Edit Preferences** (Main window) to open the Preferences dialog box.

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, with the arguments limiting the list to the specified files. 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
```

- Display all simulator state variables. (Displays only the variables that relate to the design being simulated):

report simulator state

Returns:

```
#now = 0.0
#delta = 0
#library = work
#entity = type_clocks
#architecture = full
#resolution = 1ns
```

Related Topics

[modelsim.ini Variables \[ModelSim User's Manual\]](#)

[“Setting GUI Preferences” \[ModelSim GUI Reference Manual\]](#)

restart

Reloads the design elements and resets the simulation time to zero.

Syntax

```
restart [-force] [-nobreakpoint] [-nolist] [-nolog] [-nowave]
```

Description

Invoking the restart command produces the following behavior:

- If no design is loaded, the command produces a message to that effect and takes no further action.
- If a simulation is loaded, the command restarts the simulation.
- If multiple datasets are open, including a simulation, the environment changes to the simulation context, and the simulation restarts.

The restart command reloads only design elements that have changed. (SDF files are always reread during a restart.)

Shared libraries are handled as follows during a restart:

- Shared libraries that implement only VHDL foreign architectures 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 set the `DefaultRestartOptions` variable in the *modelsim.ini* file to configure defaults for the command. Refer to “[Restart Command Defaults](#)” in the User’s Manual.

To handle restarts with Verilog PLI applications, you must define a Verilog user-defined task or function, and register a miscf class of callback.

Refer to “[Verilog Interfaces to C](#)” in the User’s Manual for more information on the Verilog HDL interfaces.

Arguments

- `-force`

(optional) Specifies to restart the simulation without requiring confirmation in a popup window.

- **-nobreakpoint**
(optional) Specifies to remove all breakpoints when restarting the simulation. The default is to reinstall all breakpoints.
- **-nolist**
(optional) Specifies not to maintain the current List window environment after restarting the simulation. The default is to maintain all currently listed HDL objects and their formats.
- **-nolog**
(optional) Specifies not to maintain the current logging environment after restarting the simulation. The default is to continue logging all currently logged objects.
- **-nowave**
(optional) Specifies not to maintain the current Wave window environment after restarting the simulation. The default is for all objects displayed in the Wave window to remain in the window with the same format.

resume

Resumes execution of a macro (DO) file after a pause command or a breakpoint.

Syntax

resume

Description

You can enter this command manually, or place it in an [onbreak](#) command string. (Placing a resume command in a [bp](#) command string does not have the desired effect.) You can also use the resume command in an [onerror](#) command string, to allow an error message to print without halting the execution of the macro file.

Arguments

None

Related Topics

[pause](#)

[Useful Commands for Handling Breakpoints and Errors \[ModelSim User's Manual\]](#)

run

Advances the simulation by the specified number of timesteps.

Syntax

```
run {[<timesteps>[<time_units>]] | -all | -continue | -init | -next} | {-step [-current] [<n>] [-out] [-over [<n>]] [-this]}
```

Description

You can use the following preference variables to control any return values after the run operation completes:

- noRunTimeMsg — Set this variable to 0 to display simulation time and delta information, or set it to 1 to disable the display of this information.
- noRunStatusMsg — Set this variable to 0 to display run status information, or set it to 1 to disable the display of this information.

The following example shows a series of run commands, and how the output changes with the preference variable settings:

```
VSIM 1> run 105
VSIM 2> set PrefMain(noRunTimeMsg) 0
# 0

VSIM 3> run 112
# Time: @217 ns 0

VSIM 4> set PrefMain(noRunStatusMsg) 0
# 0

VSIM 5> run 100
# Time: @317 ns 0
# Status: ready end

VSIM 6> set PrefMain(noRunTimeMsg) 1
# 1

VSIM 7> run 50
# Status: ready end

VSIM 8> set PrefMain(noRunStatusMsg) 1
# 1

VSIM 9> run 55

VSIM 10>
```

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 with the RunLength and UserTimeUnit *modelsim.ini* file variables. (Refer to [RunLength](#) and [UserTimeUnit](#) in the User's Manual.)

- <timesteps>[<time_units>]

(optional) Specifies the number of timesteps for the simulation to run. The number can be fractional, or can be specified as absolute by preceding the value with the character @.

<time_units> — Any valid time unit: fs, ps, ns, us, ms, or sec. 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 can be input manually, or can be used as the last command in a [bp](#) command string.

- -final

(optional) Instructs the simulator to run all final blocks, then exit the simulation.

- -init

(optional) Initializes non-trivial static SystemVerilog variables before beginning the simulation; for example, expressions involving other variables and function calls. This can be useful 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 have been initialized by that point.

- -next

(optional) Causes the simulator to run to the next event time.

- -step

(optional) Steps the simulator to the next HDL.

You can observe current values of local HDL variables in the Locals window at this time. You can specify the following arguments when you use -step:

-current

(optional) Instructs the simulation to step into an instance, process, or thread and stay in the current thread. Prevents stepping into a different thread.

<n>

(optional) Moves the simulator <n> steps ahead. Moves the debugger <n> lines ahead when you are using C Debug. Specified as a positive integer value.

-out

(optional) Instructs the simulation to step out of the current function or procedure and return to the caller.

-over

(optional) Directs ModelSim to run VHDL procedures and functions and Verilog tasks and functions, but to treat them as simple statements instead of entering and tracing them line by line.

You can use the **-over** argument to skip over VHDL procedures or functions and Verilog tasks or functions.

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 method of a SystemVerilog class when “this” refers to the specified class handle. To obtain the handle of the class, use the [examine -handle](#) command.

<class_handle> — Specifies a SystemVerilog class. Note that you must use quotation marks (" ") with this argument.

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 -current /top/p

Related Topics

[Simulate Toolbar \[ModelSim GUI Reference Manual\]](#)

[step](#)

runStatus

Returns the current state of your simulation to stdout after a run or step command.

Syntax

runStatus [[-full](#)]

Arguments

- [-full](#)

(optional) Appends additional information to the output of the runStatus command.

Return Values

[Table 2-4](#) (runStatus Command States) and [Table 2-5](#) (runStatus -full Command Information) 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 (cont.)

-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 stop command or equivalent.
gate_oscillation	Verilog gate iteration limit reached.
simulation_stop	pli stop_simulation() called.

searchlog

Searches one or more of the currently open logfiles for a specified condition.

Syntax

```
searchlog [-command <cmd>] [-count <n>] [-deltas] [-endtime <time> [<unit>]] [-env <path>]
           [-event <time>] [-expr {<expr>}] [-reverse] [-rising | -falling | -anyedge]
           [-startDelta <num>] [-value <string>] <startTime> [<unit>] <pattern>
```

Description

You can use this command to search for rising or falling edges, for signals equal to a specified value, or for when a generalized expression becomes true. If the search finds at least one match, the search 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 you specify the -deltas option, the delta of the last match is also returned:

```
{ {<time>} <delta> <matchCount> }
```

If the search does not find any matches, it returns a **TCL_ERROR**. Finding one or more matches, but less than the requested number, is not considered an error condition, and the time of the farthest match is returned, with the count of the matches found.

Arguments

- **-command <cmd>**

(optional) Specifies a Tcl command to be called for each event on the specified signal.

<cmd> — A Tcl command that receives four arguments and returns one of three values: "continue", "stop", or "" (empty).

The command is passed four arguments: the reason for the call, the time of the event, the delta for the event, and the value. The reason value is one of WLF_STARTLOG, WLF_ENDLOG, WLF_EVENT, or WLF_WAKEUP. The function is expected to return "continue", "stop", or "" (empty). If "continue" or "" (empty) is returned, the search continues, making additional callbacks as necessary. If "stop" is returned, the search stops and control returns to the caller of the searchlog command.

The command supports searching for only a single signal.

- **-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 end the search. By default, no end time is specified. When searching backwards, you must specify this option after **-reverse**.
 <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. The default is to specify the current simulation resolution by omitting <unit>. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}). Valid VHDL time units are: fs, ps, ns, us, ms, sec, min, and hr.
For -reverse searches, the specified end time must be earlier than the specified <startTime>. For -reverse searches the default <endTime> is 0.
- **-env <path>**
(optional) Specifies a path to a design region. You cannot use wildcards.
- **-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 searches until the expression evaluates to true.
 {<expr>} — An expression that evaluates to a boolean true. See “[GUI_expression_format](#)” on page 33 for the format of the expression.
- **-reverse**
(optional) Specifies to search backwards in time from <startTime>. You can limit the time span for the reverse search by including the -endtime <time> argument.
- **-rising**
(optional) Specifies a search for rising edge on a scalar signal. Ignored for compound signals.
- **-falling**
(optional) Specifies a search for falling edge on a scalar signal. Ignored for compound signals.
- **-anyedge**
(optional) Specifies a search for a rising or falling edge on a scalar signal. 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. The default is to specify the current simulation resolution by omitting <unit>. If <unit> is specified, you must enclose <time> and <unit> within curly braces ({}). Valid VHDL time units are: fs, ps, ns, us, ms, sec, min, and hr.
- **<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 the final argument to the searchlog command.

See

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, displays five lines 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.

Examples

- Display 2 lines before and 5lines after the current execution line.

see 2 5

```
# 92 :  
# 93 :           if (verbose) $display("Read/Write test done");  
# ->94 :           $stop(1);  
# 95 :           end  
# 96 :           end  
# 97 :  
# 98 :           or2 i1 (  
# 99 :           .y(t_set),
```

setenv

Changes or reports the current value of an environment variable.

Note

 The setting is valid only for the current ModelSim session.

Syntax

`setenv <varname> [<value>]`

Arguments

- `<varname>`
(required) The name of the environment variable to set or check. Must be 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

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.

Syntax

`shift`

Description

The shift command and macro parameters are used in macro files. When a macro file requires more than nine parameters, you can use the shift command to access them.

To determine the current number of macro parameters, use the `argc` simulator state variable. Refer to “[Simulator State Variables](#)” in the User’s Manual for more information.

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.

Arguments

None

show

Lists HDL objects and subregions visible from the current environment.

Syntax

```
show [-all] [<pathname>]
```

Description

The list of objects includes:

- VHDL — signals, processes, constants, variables, and instances.
- Verilog — nets, registers, tasks, functions, instances, variables, and memories.

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.

Arguments

- **-all**
(optional) Displays all names at and below the specified path recursively.
- **<pathname>**
(optional) Specifies the pathname of the environment for which to list the objects and subregions; if omitted, the current environment is assumed.

Examples

- List the names of all objects and subregion environments visible in the current environment.

show

- List the names of all objects and subregions visible in the environment named /uut.

show /uut

- List the names of all objects and subregions visible in the environment named sub_region which is directly visible in the current environment.

show sub_region

- List the names of all objects and subregions visible in all top-level environments.

show -all /

simstats

Returns performance-related statistics about elaboration and simulation with the data for each statistic on a separate line.

Syntax

```
simstats [elabcpu | elabmemory | elabtime | license | logcpu | logtime | simcpu | simmemory |  
simtime | tlcmdcpu | tlcmdtime | totalcpu | totaltime | verbose] [kb]
```

Description

The statistics measure the simulation kernel process 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 statistics and their related units on separate lines. For example:

```
# Memory Statistics  
#     mem: size after elab (VSZ)          88.89 Mb  
#     mem: size during sim (VSZ)           97.17 Mb  
# Elaboration Time  
#     elab: wall time                   0.41 s  
#     elab: cpu time                  0.23 s  
# Simulation Time  
#     sim: wall time                 1.18 s  
#     sim: cpu time                  0.66 s  
# Tcl Command Time  
#     cmd: wall time                356.39 s  
#     cmd: cpu time                  0.80 s  
# Total Time  
#     total: wall time              357.98 s  
#     total: cpu time             1.68 s
```

You can use the [simstatslist](#) command to provide this output as a continuous display (without line breaks).

All statistics are measured at the time you invoke simstats. See the arguments below for descriptions of each statistic.

Units for time values are in seconds. Units for memory values are auto-scaled.

Note

 Different operating systems report these numbers differently.

Arguments

- elabcpu
 - (optional) Returns cpu time consumed by vsim elaboration.
- elabmemory
 - (optional) Returns memory consumed during vsim elaboration.

- **elabtime**
(optional) Returns wall clock time consumed by vsim elaboration.
- **kb**
(optional) Returns statistics in kilobyte units with no auto-scaling.
- **license**
(optional) Returns a ‘License Statistics’ section that includes license statistics for checkout time and checked-out license feature names.
- **logcpu**
(optional) Returns cpu time consumed by WLF logging.
- **logtime**
(optional) Returns wall clock time consumed by WLF logging.
- **simcpu**
(optional) Returns cumulative cpu time consumed by all run commands
- **simmemory**
(optional) Returns memory consumed during the whole simulation, including the elaboration memory.
- **simtime**
(optional) Returns cumulative wall clock time consumed by all run commands.
- **tclcmdcpu**
(optional) Returns cpu time consumed by all TCL commands, excluding run commands.
- **tclcmdtime**
(optional) Returns wall clock time consumed by all TCL commands, excluding run commands.
- **totalcpu**
(optional) Returns total cpu time consumed by vsim command.
- **totaltime**
(optional) Returns total wall clock time consumed by vsim command.
- **verbose**
(optional) Displays verbose performance statistics, including an ‘elab’ report for checked-out license feature names.

simstatslist

Returns performance-related statistics about elaboration and simulation as a continuous list (without line breaks).

Syntax

```
simstatslist [elabcpu | elabmemory | elabtime | logcpu | logtime | simcpu | simmemory | simtime  
| tclcmdcpu | tclcmdtime | totalcpu | totaltime]
```

Description

Use this command in place of the [simstats](#) command to display the original statistics in a continuous format (without line breaks). For example:

```
 {{elab memory} 105348} {{sim memory} 171492} {{elab time} 0.410009}  
 {{elab cpu time} 0.234002} {{sim time} 1.18003} {{sim cpu time} 0.655204}  
 {{tclcmd time} 411.601} {{tclcmd cpu time} 0.795605}  
 {{total time} 413.191} {{total cpu time} 1.68481}
```

All statistics are measured at the time you invoke simstatslist. See the arguments below for a description of each statistic.

Units for time values are in seconds. Units for memory values are in kilobytes.

Note

 Different operating systems report these numbers differently.

Arguments

- elabcpu
 - (optional) Returns cpu time consumed by vsim elaboration.
- elabmemory
 - (optional) Returns memory consumed during vsim elaboration.
- elabtime
 - (optional) Returns wall clock time consumed by vsim elaboration.
- logcpu
 - (optional) Returns cpu time consumed by WLF logging.
- logtime
 - (optional) Returns wall clock time consumed by WLF logging.
- simcpu
 - (optional) Returns cumulative cpu time consumed by all run commands

- **simmemory**
(optional) Returns memory consumed during the whole simulation, including the elaboration memory.
- **simtime**
(optional) Returns cumulative wall clock time consumed by all run commands.
- **tclcmdcpu**
(optional) Returns cpu time consumed by all TCL commands, excluding run commands.
- **tclcmdtime**
(optional) Returns wall clock time consumed by all TCL commands, excluding run commands.
- **totalcpu**
(optional) Returns total cpu time consumed by vsim command.
- **totaltime**
(optional) Returns total wall clock time consumed by vsim command

stack down

Moves down the call stack.

Syntax

stack down [n]

Description

If invoked without arguments, the command moves down the call stack by 1 level. The Locals window displays local variables at the level.

Arguments

- n

(optional) Moves down the call stack by n levels. The default value is 1.

Related Topics

[stack frame](#)

[stack level](#)

[stack tb](#)

[stack up](#)

stack frame

Selects the specified call frame.

Syntax

stack frame n

Arguments

- <n>

Selects call frame number n. The currently executing frame is zero (also called the innermost) frame, frame one is the frame that called the innermost, and so on. The highest numbered frame is that of main.

Related Topics

[stack down](#)

[stack level](#)

[stack tb](#)

[stack up](#)

stack level

Reports the current call frame number.

Syntax

stack level

Arguments

None

Related Topics

[stack down](#)

[stack frame](#)

[stack tb](#)

[stack up](#)

stack tb

Displays a stack trace in the Transcript window, listing the sequence of HDL function calls that have been entered to arrive at the current state for the active process.

Note

 The tb command is an alias for the stack tb command.

Syntax

tb

Description

None

Arguments

None

stack up

Moves up the call stack.

Syntax

stack up [n]

Description

If invoked without arguments, the command moves up the call stack by 1 level. The Locals window displays local variables at the level.

Arguments

- n

(optional) Moves up the call stack by n levels. The default value is 1 level.

Related Topics

[stack down](#)

[stack frame](#)

[stack level](#)

[stack tb](#)

status

Lists summary information about currently interrupted macros.

Syntax

status [[file](#) | [line](#)]

Description

If invoked without arguments, the command returns the filename and line number of each interrupted macro, as well as information about which command within the macro was interrupted. It also displays any [onbreak](#) or [onerror](#) commands that have been defined for each interrupted macro.

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

[pause](#)

[resume](#)

step

An alias for the run command with the -step switch. Steps the simulator to the next HDL statement, enabling you to observe current values of local HDL variables in the Locals window.

Syntax

```
step [-current] [<n>] [-out] [-over [<n>]] [-this "this==<class_handle>"]
```

Description

You can control any return values after the step operation completes with the following preference variables:

- noRunTimeMsg — Set this variable to 0 to display simulation time and delta information or set it to 1 to disable the display of this information.
- noRunStatusMsg — Set this variable to 0 to display run status information or set it to 1 to disable the display of this information.

The following example shows a series of run commands (the step command behaves similarly) and how the output changes with the preference variable settings:

```
VSIM 1> run 105
VSIM 2> set PrefMain(noRunTimeMsg) 0
# 0

VSIM 3> run 112
# Time: @217 ns 0

VSIM 4> set PrefMain(noRunStatusMsg) 0
# 0

VSIM 5> run 100
# Time: @317 ns 0
# Status: ready end

VSIM 6> set PrefMain(noRunTimeMsg) 1
# 1

VSIM 7> run 50
# Status: ready end

VSIM 8> set PrefMain(noRunStatusMsg) 1
# 1

VSIM 9> run 55

VSIM 10>
```

Arguments

- **-current**
(optional) Instructs the simulation to step into an instance, process, or thread and stay in the current thread. Prevents stepping into a different thread.
- **<n>**
Moves the simulator <n> steps ahead. Specified as a positive integer value.
- **-out**
(optional) Instructs the simulation to step out of the current function or procedure and return to the caller.
- **-over**
(optional) Directs ModelSim to run VHDL procedures and functions, Verilog tasks and functions but to treat them as simple statements instead of entering and tracing them line by line.

You can use the -over argument to skip over a VHDL procedure or function, Verilog task or 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 method of a SystemVerilog class when "this" refers to the specified class handle. To obtain the handle of the class, use the [examine -handle](#) command.
<class_handle> — Specifies a SystemVerilog class. Note that you must use quotation marks (" ") with this argument.

Related Topics

[run](#)

[Stepping Through Your Design \[ModelSim User's Manual\]](#)

stop

Used with the when command to stop simulation in batch files.

Syntax

stop [-sync]

Description

The stop command has the same effect as hitting a breakpoint. You can place the stop command anywhere within the body of the when command.

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 use a [when](#) command to stop the simulation, 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.

Arguments

- -sync
(optional) Stops the currently running simulation at the next time step.

Related Topics

[resume](#)

[run](#)

suppress

Prevents one or more specified messages from displaying.

Note

 To use the suppress command, you must have a design loaded in ModelSim. Otherwise, ModelSim will display an error message without running the command.

Syntax

```
suppress [-clear <msg_number>[,<msg_number>,...]] [<msg_number>[,<msg_number>,...]]  
[<code_string>[,<code_string>,...]]
```

Description

You cannot suppress Fatal or Internal messages. If you enter the suppress command without arguments, it returns the message numbers of all suppressed messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to [suppress](#) and “[Message Severity Level](#)” in the User’s Manual for more information.

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. Use a comma-separated list to specify multiple message numbers.
- `<msg_number>[,<msg_number>,...]`
(optional) A number identifying the message to be suppressed. Use a comma-separated list to specify multiple message numbers.
- `<code_string>[,<code_string>,...]`
(optional) A string identifier of the message to be suppressed. Disables warning messages in the category specified by `<code_string>`. Warnings that can be disabled include the `<code_string>` 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
```

tb

Displays a stack trace for the current process in the Transcript window.

Syntax

`tb`

Description

The `tb` (traceback) command lists the sequence of HDL function calls that have been entered to arrive at the current state for the active process. The `tb` command is an alias for the stack `tb` command.

Arguments

None

Time

Used as the suffix for a collection of related commands that allow you to perform comparisons between, operations on, and conversions of, time values for simulation.

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.

Useful when an integer calculation results in a 64-bit value that you need to convert to a time unit.

`scaleTime <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) indicating whether the given string is a valid time for use with any of these Time calculations.

`formatTime {+|-} commas | {+|-}nodefunit | {+|-}bestunits`

Sets display properties for time values.

Description

When you do not specify [unit], each of these commands assumes the current simulation time unit, as specified by the Resolution variable in the *modelsim.ini* file, or by the `vsim -t` command. (Refer to the User's Manual for more information on the **Resolution** variable.) For most commands, you can specify units of time (such as ns, us, ps) independently for each <time[1 | 2]>. See the description of each command and examples for more information.

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. 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> — The real number to use as scaling factor. Common values can include:

0.25, 0.5, 1.5, 2, 10, 100

- {+ | -} commas — Controls whether commas display 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

- Entering the command:

```
>ltTime 100ns 1ms
```

Returns:

```
# 1
```

- Entering the command:

```
>addTime {1545 ns} {455 ns}
```

Returns:

```
# 2 us
```

- Entering the command:

```
>gteTime "1000 ns" "1 us"
```

Returns:

```
# 1
```

- Entering the command:

```
>divTime 1us 10ns
```

Returns:

```
# 100
```

- Entering the command:

```
>formatTime +bestunit
```

Returns:

```
# -commas -nodefunit +bestunits
```

- Entering the command:

>scaleTime 3ms 1000

Returns:

3 sec

- Entering the command:

>RealToTime 1.345e04

Returns:

13450 ns

transcript

Controls echoing of commands executed in a macro file. Reports the current setting when entered with no options.

Syntax

transcript [on | off | -q | quietly]

Arguments

- **on**
(optional) Specifies that commands in a macro file are echoed to the Transcript window as they are executed.
- **off**
(optional) Specifies that commands in a macro file are not echoed to the Transcript window as they are executed.
- **-q**
(optional) Returns "0" if transcripting is turned off or "1" if transcripting is turned on. Useful in a Tcl conditional expression.
- **quietly**
(optional) Turns off the transcript echo for all commands. For information about turning off echoing for individual commands, see the [quietly](#) command.

Examples

- Echo commands within a macro file to the Transcript window as they are executed:

transcript on

- If issued immediately after the previous example, the command:

transcript

returns

Macro transcripting is turned ON.

Related Topics

[Transcript Window \[ModelSim GUI Reference Manual\]](#)

transcript file

Sets or queries the current PrefMain(file) Tcl preference variable.

Syntax

transcript file [<filename>]

Arguments

- <filename>

(optional) Specifies a name for the transcript file. Allows wildcard characters, and “stdout” or “stderr” are valid file names. Specifying a new file closes the existing transcript file, and opens a new transcript file. Specifying an empty string (“”) closes the existing file, and no new file is opened. If you do not specify this argument, the current filename is returned.

Note

 To prevent overwriting older transcript files, include 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.

Description

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.

Examples

- Close the current transcript file and stop 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*.

- Create a transcript containing only data from the second millisecond of the simulation.

```
transcript file ""
run 1 ms
transcript file transcript
run 1 ms
```

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.

Related Topics

- ["Creating a Transcript File" \[ModelSim User's Manual\]](#)
- [“Setting GUI Preferences” \[ModelSim GUI Reference Manual\]](#)
- [Transcript Window \[ModelSim GUI Reference Manual\]](#)
- [transcript path](#)
- [transcript sizelimit](#)

transcript path

Returns the full pathname to the current transcript file.

Syntax

`transcript path`

Arguments

None

Related Topics

["Creating a Transcript File" \[ModelSim User's Manual\]](#)

[“Setting GUI Preferences” \[ModelSim GUI Reference Manual\]](#)

[“Transcript Window” \[ModelSim GUI Reference Manual\]](#)

[transcript file](#)

transcript sizelimit

Sets or queries the current preference value for the transcript fileSizeLimit value.

Syntax

`transcript sizelimit [<size>]`

Arguments

- `<size>`

(optional) Specifies the size, in KB, of the transcript file. The default is 0 or unlimited. The actual file size can 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

 If the size limit is reached, the transcript file is saved and simulation continues. You can set the size of the transcript file with the \$PrefMain (fileSizeLimit) Tcl variable in the Preferences dialog. Refer to “[Setting GUI Preferences](#)” in the GUI Reference Manual for more information.

Related Topics

["Creating a Transcript File"](#) [ModelSim User's Manual]

[“Setting GUI Preferences”](#) [ModelSim GUI Reference Manual]

[“Transcript Window”](#) [ModelSim GUI Reference Manual]

[transcript file](#)

transcript wrapcolumn

Defines the column width when wrapping output lines in the transcript file.

Syntax

```
transcript wrapcolumn <integer>
```

Arguments

- <integer>

An integer that defines the width, in characters, before forcing a line break. The default value is 30000.

Description

The wrap occurs at the first white-space character after reaching the transcript wrapwscolumn value, or at exactly the column width if no white-space is found.

transcript wrapmode

Controls wrapping of output lines in the transcript file.

Syntax

`transcript wrapmode [0 | 1 | 2]`

Arguments

- 0
(default) Disables wrapping.
- 1
Enables wrapping, based on the value of the transcript wrapcolumn command, which defaults to 30,000 characters.
- 2
Enables wrapping and adds a continuation character (\) at the end of every wrapped line, except for the last.

transcript wrapwscolumn

Defines the column width when wrapping output lines in the transcript file.

Usage

```
transcript wrapwscolumn <integer>
```

Arguments

- <integer>

An integer specifying that the wrap will occur at the first white-space character after reaching the specified number of characters. If there is no white-space, the wrap occurs at the transcript wrapcolumn value. The default value is 27000.

tssi2mti

Converts a vector file in TSSI Format into a sequence of force and run commands.

Syntax

```
tssi2mti <signal_definition_file> [<sef_vector_file>]
```

Description

This command writes the stimulus to the standard output.

The source code for tssi2mti is provided in the examples directory as:

```
<install_dir>/examples/tssi2mti/tssi2mti.c
```

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 convert. If no file is specified, standard input is used.

Examples

- The following command produces 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

[run](#)

ui_VVMode

Specifies the actions to take when encountering UI registration calls used by verification packages. Returns the current setting when specified without an argument.

Syntax

```
ui_VVMode [full | logclass | logobj | nolog | off ]
```

Description

UI registration calls, which are Verilog system tasks specific to ModelSim, are typically included in verification packages to make key information about the packages 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.

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.

unsetenv

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 to delete.

Related Topics

[setenv](#)

[printenv](#)

vcd add

Adds the specified objects to a VCD file.

Syntax

```
vcd add [-dumports] [-file <filename>] [[-in] [-out] [-inout] | [-ports]] [-internal] [-l <level>] [-nocell] [-r | -r-optcells] <object_name> ...
```

Description

The command allows you to add the following objects:

- Verilog nets and variables
- VHDL signals: bit, bit_vector, std_logic, std_logic_vector, integer, vector of integers, signed vector, unsigned vector, positive, natural (other types are silently ignored).

The command works with mixed HDL designs.

All vcd add commands must execute at the same simulation time. The command adds the objects you specify to the VCD header and records their subsequent value changes in the specified VCD file. By default, the file captures all port driver changes and internal variable changes. You can use the command arguments to filter the output.

Related Verilog tasks: \$dumpvars, \$fdumpvars

Arguments

- **-dumports**
(optional) Specifies port driver changes to add to an extended VCD file. When the [vcd dumpports](#) command cannot specify all port driver changes that will appear in the VCD file, you can use multiple vcd add -dumports commands to specify additional port driver changes.
- **-file <filename>**
(optional) Specifies the name of the VCD file. Use this option only when you have used the [vcd files](#) command to create multiple VCD files.
 <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.
- **-l <level>**
(optional) Specifies the number of hierarchical levels to be included in the logging process.
- **-nocell**
(optional) Suppresses the logging of signals within a cell.
- **-r | -r -optcells**
(optional) Specifies that signal and port selection occurs recursively into subregions. Omitting this option limits included signals and ports to the current region. Using **-r** with **-optcells** 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.
- **<object_name> ...**
(required) Specifies the Verilog or VHDL object or objects to add to the VCD file. You can specify multiple objects by separating names with spaces. Accepts wildcards. Must be the final argument to the vcd add command.

Related Topics

[vcd checkpoint](#)

[vcd comment](#)

[vcd dumpsupports](#)

[vcd dumpsupportsall](#)

[vcd dumpsupportsflush](#)

[vcd dumpportslimit](#)

[vcd dumpportsoff](#)

[vcd dumpportson](#)

[vcd file](#)

[vcd files](#)

[vcd flush](#)

[vcd limit](#)

[vcd off](#)

[vcd on](#)

[vcd2wlf](#)

[DumpportsCollapse \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd checkpoint

Dumps the current values of all VCD variables to the specified VCD file.

Syntax

`vcd checkpoint [<filename>]`

Arguments

- `<filename>`

(optional) Specifies the name of the VCD file. If you omit `<filename>`, the command executes on the file designated by either the [vcd file](#) command or *dump.vcd* if vcd file was not invoked.

Description

While simulating, dumps only VCD variable value changes. Related Verilog tasks are \$dumpall and \$fdumpall.

Related Topics

[vcd add](#)
[vcd comment](#)
[vcd dumports](#)
[vcd dumportsall](#)
[vcd dumportsflush](#)
[vcd dumportslimit](#)
[vcd dumpportsoff](#)
[vcd dumpportson](#)
[vcd file](#)
[vcd files](#)
[vcd flush](#)
[vcd limit](#)
[vcd off](#)
[vcd on](#)
[vcd2wlf](#)

[DumportsCollapse \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd comment

Inserts the specified comment in the specified VCD file.

Syntax

`vcd comment <comment string> [<filename>]`

Arguments

- `<comment string>`
(required) Comment to include in the VCD file. You must enclose the comment in double quotation marks or curly braces. Must be the first argument to the vcd comment command.
- `<filename>`
(optional) Specifies the name of the VCD file. If you omit `<filename>`, the command executes either on the file designated by the [vcd file](#) command, or on *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 \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd dumports

Creates a VCD file that includes port driver data.

Syntax

```
vcd dumports [-compress] [-direction] [-file <filename>] [-force_direction] [-in] [-out] [-inout]  
[-no_strength_range] [-unique] [-vcdstim] <object_name> ...
```

Description

By default all port driver changes are captured in the file. You can use the arguments detailed below to filter the output. Related Verilog task: \$dumports.

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. This does not create an eVCD file.
- **-file <filename>**
(optional) Creates a VCD file. Defaults to the current working directory and the filename *dumports.vcd*. You can open multiple filenames during a single simulation.
 <filename> — Specifies a filename. Specify with a .gz extension to compress the file.
- **-force_direction**
(optional) Causes vcd dumports 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](#) in the User's Manual for additional information.

- **-unique**
(optional) Generates unique VCD variable names for ports, even if those ports connect 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](#) in the User's Manual for further information.
- **<object_name> ...**
(required) Specifies one or more HDL objects to add to the VCD file. Separate names with spaces to specify multiple objects. Accepts wildcards. Must be 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](#) in the User's Manual 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 resimulates the design using the VCD files in place of the instance source files. Refer to [Replacing Instances with Output Values from a VCD File](#) in the User's Manual 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](#)

[DumportsCollapse \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd dumportsall

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.

Note

The related Verilog task is \$dumportsall.

Syntax

`vcd dumportsall [<filename>]`

Arguments

- `<filename>`
(optional) Specifies the name of the VCD file. If you omit `<filename>`, the command executes on all open VCD files.

Related Topics

[vcd add](#)
[vcd checkpoint](#)
[vcd comment](#)
[vcd dumports](#)
[vcd dumportsflush](#)
[vcd dumportslimit](#)
[vcd dumpportsoff](#)
[vcd dumpportson](#)
[vcd file](#)
[vcd files](#)
[vcd flush](#)
[vcd limit](#)
[vcd off](#)
[vcd on](#)
[vcd2wlf](#)

[DumportsCollapse \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd dumpportsflush

Flushes the contents of the VCD file buffer to the specified VCD file.

Note

 The related Verilog task is \$dumpportsflush.

Syntax

`vcd dumpportsflush [<filename>]`

Arguments

- `<filename>`

(optional) Specifies the name of the VCD file. If you omit `<filename>`, the command executes 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 \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd dumpportslimit

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.

Syntax

vcd dumpportslimit <dumplimit> [<filename>]

Description

Related Verilog task: \$dumpportslimit

Arguments

- <dumplimit>
(required) Specifies the maximum VCD file size in bytes. Must be the first argument to the vcd dumpportslimit command.
- <filename>
(optional) Specifies the name of the VCD file. If you omit <filename>, the command executes 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 \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd dumpportoff

Turns off VCD dumping and records all dumped port values as x.

Syntax

vcd dumpportoff [<filename>]

Description

Related Verilog task: \$dumpportoff

Arguments

- <filename>

(optional) Specifies the name of the VCD file. If you omit <filename>, the command executes 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 \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd dumpportson

Turns on VCD dumping and records the current values of all selected ports. Typically used to resume dumping after invoking vcd dumpportsoff.

Note

 The related Verilog task is \$dumpportson.

Syntax

vcd dumpportson [<filename>]

Arguments

- <filename>
(optional) Specifies the name of the VCD file. If you omit <filename>, the command executes 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 \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd file

Specifies the filename and state mapping for the VCD file created by a vcd add command.

Note



Syntax

```
vcd file [-dumports] [-direction] [<filename>] [-map <mapping pairs>] [-no_strength_range]  
[-nomap] [-unique]
```

Description

The vcd file command is optional. If you use it, you must issue it before any vcd add commands.

Arguments

- **-dumports**
(optional) Captures 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. Does not create an eVCD file.
- **<filename>**
(optional) Specifies the name of the created VCD file. 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 is the character you want to record 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](#) in the User's Manual for additional information.
- **-nomap**
(optional) Affects only VHDL signals of type std_logic. Specifies to use the std_logic enumeration characters of UX01ZWLH- for values recorded in the VCD file. 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 connect 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 \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd files

Specifies filenames and state mapping for VCD files created by the vcd add command.

Note

 The related Verilog task is \$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 do not use the -compress argument.
- **-direction**
(optional) Includes driver direction data in the VCD file. Does not create an eVCD 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 is the character you wish to record 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](#)” in the User’s Manual for additional information.
- **-nomap**
(optional) Affects only VHDL signals of type std_logic. Specifies to use the std_logic enumeration characters of UX01ZWLH- for values recorded in the VCD file. 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 connect to the same collapsed net.

Description

The vcd files command is optional. If you use the command, you must issue it before any vcd add commands.

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 dumports](#)
[vcd dumportsall](#)
[vcd dumportsflush](#)
[vcd dumportslimit](#)
[vcd dumpportsoff](#)
[vcd dumpportson](#)
[vcd file](#)

[vcd flush](#)

[vcd limit](#)

[vcd off](#)

[vcd on](#)

[vcd2wlf](#)

[DumpportsCollapse \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd flush

Flushes the contents of the VCD file buffer to the specified VCD file. Related Verilog tasks:
\$dumpflush, \$fdumpflush

Note

 The related Verilog tasks are \$dumpflush and \$fdumpflush.

Syntax

vcd flush [<filename>]

Arguments

- <filename>
(optional) Specifies the name of the VCD file. If you omit <filename>, the command executes on the file designated by the [vcd file](#) command, or on *dump.vcd* if vcd file was not invoked.

Description

Use this command when you want to create a complete VCD file without ending your current simulation.

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 \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd limit

Specifies the maximum size of a VCD file (by default, limited to available disk space).

Note

 The related Verilog tasks are \$dumplimit and \$fdumplimit.

Syntax

```
vcd limit <filesize> [<filename>]
```

Description

When the size of the file exceeds the limit, a comment is appended to the file and VCD dumping is disabled.

Arguments

- <filesize>

(Required) Specifies the maximum VCD file size, in bytes. The numerical value of <filesize> must be a whole number. Must be 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 you omit <filename>, the command executes on the file designated by the [vcd file](#) command, or on *dump.vcd* if vcd file was not invoked.

Examples

- 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 dumports](#)
[vcd dumportsall](#)
[vcd dumportsflush](#)
[vcd dumportslimit](#)

[vcd dumpportoff](#)

[vcd dumpportson](#)

[vcd file](#)

[vcd files](#)

[vcd flush](#)

[vcd off](#)

[vcd on](#)

[vcd2wlf](#)

[DumpportsCollapse \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd off

Turns off VCD dumping to the specified file and records all VCD variable values as x.

Note

 The related Verilog tasks are \$dumpoff and \$fdumpoff.

Syntax

vcd off [<filename>]

Arguments

- <filename>

(optional) Specifies the name of the VCD file. If you omit <filename>, the command executes on the file designated by the [vcd file](#) command, or on *dump.vcd* if vcd file was not invoked.

Related Topics

[vcd add](#)
[vcd checkpoint](#)
[vcd comment](#)
[vcd dumports](#)
[vcd dumportsall](#)
[vcd dumportsflush](#)
[vcd dumportslimit](#)
[vcd dumpportsoff](#)
[vcd dumpportson](#)
[vcd file](#)
[vcd files](#)
[vcd flush](#)
[vcd limit](#)
[vcd on](#)
[vcd2wlf](#)

[DumportsCollapse \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcd on

Turns on VCD dumping to the specified file and records the current values of all VCD variables.

Syntax

`vcd on [<filename>]`

Description

By default, vcd on runs automatically at the end of any simulation time invoked by the [vcd add](#) command.

Related Verilog tasks: \$dumpon, \$fdumpon

Arguments

- `<filename>`
(optional) Specifies the name of the VCD file. If you omit <filename>, the command executes on the file designated by the [vcd file](#) command, or on *dump.vcd* if vcd file was not invoked.

Related Topics

[vcd add](#)
[vcd checkpoint](#)
[vcd comment](#)
[vcd dumports](#)
[vcd dumportsall](#)
[vcd dumportsflush](#)
[vcd dumportslimit](#)
[vcd dumpportsoff](#)
[vcd dumpportson](#)
[vcd file](#)
[vcd files](#)
[vcd flush](#)
[vcd limit](#)
[vcd off](#)
[vcd2wlf](#)

[DumportsCollapse \[ModelSim User's Manual\]](#)

Value Change Dump (VCD) Files [ModelSim User's Manual]

vcd2wlf

Translates a VCD (Value Change Dump) file into a WLF file that you can display in ModelSim using the vsim -view argument. Works only on VCD files containing positive time values.

Syntax

```
vcd2wlf [-splitio] [-splitio_in_ext <extension>] [-splitio_out_ext <extension>] [-nocase]
          {<vcd filename> | - } <wlf filename>
```

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. To ensure that bits get converted together as a bus, 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

- **-splitio**
(optional) Specifies to split extended VCD port values into their corresponding input and output components, creating two signals instead of just one in the resulting .wlf file. By default, the new input-component signal keeps the name of the original port, while the output-component signal has the original name with an appended "__o".
- **-splitio_in_ext <extension>**
(optional) Adds an extension to input-component signal names created with -splitio.
 <extension> — Specifies a string.
- **-splitio_out_ext <extension>**
(optional) Adds an extension to output-component signal names created with -splitio.
 <extension> — Specifies a string.
- **-nocase**
(optional) Converts all alphabetic identifiers to lowercase.
- **{<vcd filename> | - }**
(required) Specifies the name of the VCD file, or standard input (-), 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.

Examples

- Concatenate *my.vcd* file and pipe standard input to vcd2wlf and save output to *my.wlf* file.

cat my.vcd | vcd2wlf - my.wlf

- Redirect input from the file *my.vcd* file to vcd2wlf and save the output to *my.wlf* file.

vcd2wlf - my.wlf <my.vcd

Related Topics

[vcd add](#)
[vcd checkpoint](#)
[vcd comment](#)
[vcd dumports](#)
[vcd dumportsall](#)
[vcd dumportsflush](#)
[vcd dumportslimit](#)
[vcd dumpportsoff](#)
[vcd dumpportson](#)
[vcd file](#)
[vcd files](#)
[vcd flush](#)
[vcd limit](#)
[vcd off](#)
[vcd on](#)

[DumportsCollapse \[ModelSim User's Manual\]](#)

[Value Change Dump \(VCD\) Files \[ModelSim User's Manual\]](#)

vcom

Compiles VHDL source code into a specified working library (or to the work library by default).

Syntax

```
vcom [options] <filename> [<filename> ...]  
[options]:  
[-87 | -93 | -2002 | -2008]  
[-addpragmaprefix <prefix>] [-allowProtectedBeforeBody] [-amsstd | -noamsstd] [-autoorder]  
[-bindAtCompile] [-bindAtLoad]  
  
[-defercheck] [-deferSubpgmCheck | -noDeferSubpgmCheck] [-dirpath <pathname>]  
[-error <msg_number>[,<msg_number>,...]] [-explicit]  
[(-F | -file | -f) <filename>] [-fatal <msg_number>[,<msg_number>,...]] [-force_refresh  
    <primary> [<secondary>]]  
[-gen_xml <design_unit> <filename>]  
[-ignoredefaultbinding] [-ignorepragmaprefix <prefix>] [-ignoreStandardRealVector] [-  
    ignorevitalerrors] [-initoutcompositeparam | -noinitoutcompositeparam]  
[-just abcepx]  
[-logfile <filename> | -l <filename>] [-line <number>] [-lint] [-lower] [-lrmconfigvis]  
[-mixedsvvh [b | l | r ][i][pc]] [-modelsimini <path/modelsim.ini>] [-msglimit {error | warning |  
    all[,-<msgNumber>,...]} | none[,+<msgNumber>,...]] <msgNumber>,[msgNumber,...]] [-  
    msglimitcount <limit_value> -msglimit {error | warning | all[,-<msgNumber>,...]} |  
    none[,+<msgNumber>,...]] <msgNumber>,[msgNumber,...]]  
[-no1164] [-noaccel <package_name>][-nocasestaticerror] [-nocheck] [-nocreatelib] [-  
    nodbgsym] [-nofprangecheck] [-noFunctionInline] [-noindexcheck] [-nologo] [-  
    nonstddriverinit] [-noothersstaticerror] [-note <msg_number> [,<msg_number>, ...]] [-  
    novitalcheck] [-nowarn <category_number>]  
[-oldconfigvis] [-optionset <optionset_name>] [-outf <filename>]  
[-pedanticerrors] [-performdefaultbinding] [-preserve] [-[w]prof=<filename>] [-  
    proftick=<integer>]  
[-quiet]  
[-rangecheck | -norangecheck] [-refresh <primary> [<secondary>]]  
[-s] [-separateConfigLibrary] [-showsubprograms | -noshowsubprograms] [-skip abcepx] [-  
    skipsynthoffregion] [-smartdbgsym] [-source] [-stats [=+ | -]<feature>[, [+ | -]<mode>]]] [-  
    suppress {<msgNumber> | <msgGroup>} ,[,<msg_number> | <msgGroup>],...]]
```

```
[-version] [-vitalmemorycheck] [-vmake]  
[-warning <msg_number>[,<msg_number>,...]] [-warning error] [-work <library_name>]
```

Description

You can invoke vcom from ModelSim or from the command prompt of your operating system. You can invoke this command during simulation.

Compiled libraries change with major versions of ModelSim. When moving between major versions, you must use the vcom -refresh argument to refresh compiled libraries. You do not need to refresh libraries for minor versions (letter releases).

All arguments to the vcom command are case-sensitive. For example, -WORK and -work are not equivalent.

You can use the -help or -h switch for additional information on the command.

Arguments

- **-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. Refer to “[Differences Between Versions of VHDL](#)” in the User’s Manual for more information.
 - 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.
- **-addpragmaprefix <prefix>**
(optional) Enables recognition of pragmas with a user specified prefix. If you do not specify a prefix, pragmas are treated as comments.
All regular synthesis pragmas are honored.

<prefix> — Specifies a user defined string. The default is no string, indicated by quotation marks.
You can also set the prefix with the AddPragmaPrefix variable in the vcom section of the *modelsim.ini* file. Refer to [AddPragmaPrefix](#) in the User’s Manual for more information.
- **-allowProtectedBeforeBody**
(optional) Allows creation of a variable of a protected type prior to declaring the body.

- **-amsstd | -noamsstd**
(optional) 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. Refer to [AmsStandard](#) and [MGC_AMS_HOME](#) in the User's Manual for more information.
- **-autoorder**
(optional) An -autoorder compilation determines the proper order of VHDL design units, independent of the order that you listed the files on the command line. Compilation proceeds in a scan phase followed by a refresh phase. Without the argument, you must list VHDL files in their proper compilation order.
- **-bindAtCompile**
(optional) Forces ModelSim to perform default binding at compile time rather than at load time. You can change the permanent default by editing the BindAtCompile variable in the *modelsim.ini*. Refer to “[Default Binding](#)” and [BindAtCompile](#) in the User's Manual for more information.
- **-bindAtLoad**
(optional) Forces ModelSim to perform default binding at load time rather than at compile time. (Default)
- **-createlib[=compress]**
Creates libraries that do not currently exist.
compress — Compresses the created libraries
- **-defercheck**
(optional) Defers index checks until run time.
- **-deferSubpgmCheck**
(optional) Forces the compiler to report array indexing and length errors as warnings (instead of as errors) when it encounters them in 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.
- **-dirpath <pathname>**
(optional) Specifies the location of a working directory to store in the library to override the current working directory. This allows you hide the directory path information.

Caution

 Use of this argument is not recommended.

For example, if you use -dirpath to override the working directory information, when an end-user selects something in the design and asks to see the declaration, the ModelSim user interface will not be able to find the source files.

- **-error <msg_number>[,<msg_number>,...]**

(optional) Changes the severity level of the specified message(s) to "error." Edit the error variable in the *modelsim.ini* file to set a permanent default. Refer to [error](#) and "[Message Severity Level](#)" in the User's Manual for more information.

<msg_number> — A number identifying the message. Specify multiple message numbers as a comma-separated list.

- **-explicit**

(optional) Directs the compiler to resolve ambiguous function overloading by favoring the explicit function definition over the implicit function definition. This behavior does not match the VHDL standard, but the majority of EDA tools choose explicit operators over implicit operators, so activation of this switch makes ModelSim compatible with common industry practice.

- **(-F | -file | -f) <filename>**

(optional) -f, -file and -F each specifies an argument file with more command-line arguments, allowing you to use complex argument strings without retyping. You can nest -F, -f and -file commands. Allows gzipped input files.

With -F only: relative file names and paths within the arguments file <filename> are prefixed with the path of the arguments file when lookup with relative path fails. Refer to "[Argument Files](#)" on page 28 for more information.

- **-fatal <msg_number>[,<msg_number>,...]**

(optional) Changes the severity level of the specified message(s) to "fatal." Edit the fatal variable in the *modelsim.ini* file to set a permanent default. Refer to [fatal](#) and "[Message Severity Level](#)" in the User's Manual for more information.

<msg_number> — A number identifying the message. Specify multiple message numbers as a comma-separated list.

- **-force_refresh <primary> [<secondary>]**

(optional) Forces the refresh of all specified design units. Updates the work directory by default; use -work <library_name>, in conjunction with -force_refresh, to update a different library (for example, vcom -work <your_lib_name> -force_refresh).

<primary> [<secondary>] — Specifies the entity, package, configuration, or module to refresh.

- If <primary> is an entity — only that entity, no related architectures, is refreshed.

- If <primary> is a package — the only legal value of <secondary> is “body”, and only the package is refreshed.
- If you specify both <primary> and <secondary> — Only the <secondary> architecture is updated, not the entity.

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/...
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.

You cannot specify the <filename> argument when specifying this argument.

- **-gen_xml <design_unit> <filename>**

(optional) Produces an XML-tagged file containing the interface definition of the specified entity.

<design_unit> — The name of an entity or design unit in the Work library. Does not allow wildcards and multiple design unit names.

<filename> — A user-specified name for the file.

For example:

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.

```
vlib work
vcom counter.vhd
vcom -gen_xml counter counter.xml
```

- **-help [all | category | <option> | <command-line> | <category_list>]**

(optional) Provides online command help.

all — List all categories and options.

category — List all command categories.

<option> — Show help for the listed command option.

<command-line> — List all options for a particular category. Vcom categories are:

- General

- Analog
 - Coverage
 - Debug
 - GLS
 - Language
 - Library
 - Messages
 - Optimize
- -ignoredefaultbinding
- (optional) Instructs the compiler not to generate a default binding during compilation. You must explicitly bind all components in the design through either configuration specifications or configurations. If you do not fully specify an explicit binding, defaults for the architecture, port maps, and generic maps are used as needed. Edit the [RequireConfigForAllDefaultBinding](#) *modelsim.ini* variable to set a permanent default. Refer to [RequireConfigForAllDefaultBinding](#) and [Default Binding](#) in the User's Manual for more information.
- -ignorepragmaprefix <prefix>
- (optional) Directs vcom to ignore pragmas with the specified prefixname. All affected pragmas are treated as regular comments. Edit the [IgnorePragmaPrefix](#) *modelsim.ini* variable to set a permanent default. Refer to [IgnorePragmaPrefix](#) in the User's Manual.
- <prefix> — Specifies a user defined string.
- -ignoreStandardRealVector
- (optional) Instructs ModelSim to ignore the REAL_VECTOR declaration in package STANDARD when compiling with vcom -2008. Edit the [ignoreStandardRealVector](#) *modelsim.ini* variable to set a permanent default. Refer to [ignoreStandardRealVector](#) in the User's Manual. For more information refer to the REAL_VECTOR section in [Help > Technotes > vhdl2008migration](#).
- -ignorevitalerrors
- (optional) Directs the compiler to ignore VITAL compliance errors. 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, the assumption is that compliance checking has passed.
- -initoutcompositeparam
- (optional) Causes initialization of subprogram parameters for array and record types when the subprogram is executed in designs compiled with LRM 1076-2002 and earlier. This argument forces the output parameters to their default initial ("left") values when entering a subprogram. By default, -initoutcompositeparam is enabled for designs compiled with vcom

-2008 and later. You can also enable this by setting the `InitOutCompositeParam` variable to 1 in the *modelsim.ini* file. Refer to [InitOutCompositeParam](#) in the User's Manual.

- `-noinitoutcompositeparam`

(optional) Disables initialization of subprogram parameters for array and record types when the subprogram is executed in designs compiled with LRM 1076-2002 and earlier. By default, designs compiled with LRM 1076-2008 and later do not initialize subprogram parameters for array and record types when the subprogram is executed. You can also disable initialization of subprogram parameters for array and record types by setting the `InitOutCompositeParam` variable to 2 in the *modelsim.ini* file. Refer to [InitOutCompositeParam](#) in the User's Manual.

- `-just abcepx`

(optional) Directs the compiler to include only the following:

- a — architectures
- b — bodies
- c — configurations
- e — entities
- p — packages
- x — VHDL 2008 context declarations

You can use any combination in any order, but you must specify at least one choice if you use this switch.

- `-logfile <filename> | -l <filename>`

(optional) Generates a log file of the compile.

`-logfile <filename>` — Saves transcript data to `<filename>`. Can be abbreviated to `-l <filename>`. Overrides the default transcript file creation set with the `TranscriptFile` or `BatchTranscriptFile` *modelsim.ini* variables. (Refer to [TranscriptFile](#) or [BatchTranscriptFile](#) in the User's Manual.) You can also specify "stdout" or "stderr" for `<filename>`.

- `-line <number>`

(optional) Starts the compiler on the specified line in the VHDL source file. By default, the compiler starts at the beginning of the file.

`<number>` —

- `-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 applies only to designs compiled using -87. If you compile using -93, such cases are flagged as a warning or error, even without the -lint argument.

You can also enable lint checking with the Show_Lint variable in the *modelsim.ini* file. Refer to [Show_Lint](#) in the User's Manual.

- -lower

(optional) Forces vcom to convert uppercase letters in object identifiers to lowercase. You can also enable this by setting the PreserveCase variable to 0 in the *modelsim.ini* file. Refer to [PreserveCase](#) in the User's Manual.

- -lrmconfigvis

(optional, default) Forces vcom to use visibility rules that comply with the Language Reference Manual when processing VHDL configurations. Refer to vcom [-oldconfigvis](#) or to the description of the [oldVHDLConfigurationVisibility](#) *modelsim.ini* variable in the User's Manual, for information on processing visibility of VHDL component configurations consistent with prior releases.

- -mixedsvvh [b | l | r][i][pc]

(optional) Facilitates using VHDL packages at the SystemVerilog-VHDL boundary of a mixed-language design. When you compile a VHDL package with -mixedsvvh, you can include the package in a SystemVerilog design as if it were defined in SystemVerilog itself.

If you specify -mixedsvvh without arguments, ModelSim compiles VHDL vectors in the following ways:

- VHDL bit_vectors are treated as SystemVerilog bit vectors.
 - VHDL std_logic_vectors, std_ulogic_vectors, and vl_logic_vectors are treated as SystemVerilog logic vectors.
- 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. Can be specified together with b, l, or r, spaces are not allowed between arguments.
pc — Preserves case of all identifiers.

- **-modelsimini <path/modelsim.ini>**

(optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable.

<path/modelsim.ini> — Specifies either an absolute or relative path to the initialization file, including the file name. Use a forward slash (/) as the path separator on Windows systems.

- **-msglimit {error | warning | all[-<msgNumber>,...] | none[+<msgNumber>,...] | <msgNumber>,[msgNumber,...]}**

(optional) Limits messages to the default message limit count of five. Use the -msglimitcount argument to change the default message limit count. Specify multiple messages as a comma-separated list.

error — Stops the run when the total number of errors reaches the default count.

warning — Stops the run when the total number of warnings reaches the default count.

all — Limits all messages to the default count except those you specifically exclude. To exclude messages from the limit, supply a comma-separated list of message numbers, each preceded by a minus sign (-), for example “all,-<msgNumber1>,<msgNumber2>,...”.

none — Excludes all messages from the default count except those you specifically include. To include messages to limit to the default count, supply a comma-separated list of message numbers, each preceded by a plus sign (+), for example “none,+<msgNumber1>,+<msgNumber2>,...”.

<msgNumber>[,<msgNumber>,...] — Specifies messages to limit to the default count.

Examples:

- Stop the run when the total number of errors reaches the default count:

```
-msglimit error
```

- Stop the run when the total number of warnings reaches the default count:

```
-msglimit warning
```

- Specifically limit messages 2374 and 2385 to the default count:

```
-msglimit 2374,2385
```

- Limit all messages to the default count, except messages 2374 and 2385:

```
-msglimit all,-2374,-2385
```

- Limit only messages 2374 and 2385 to the default count:

```
-msglimit none,+2374,+2385
```

- `-msglimitcount <limit_value> -msglimit {error | warning | all[-<msgNumber>,...] | none[+<msgNumber>,...] | <msgNumber>,[msgNumber,...]}`
(optional) Specifies to either stop the run when the total number of reported errors/warnings reaches the user-defined limit value, or to limit the reporting of listed messages to the user-defined limit_value. Overrides the MsgLimitCount variable in the *modelsim.ini* file. Refer to [MsgLimitCount](#) in the User's manual.
- `-no1164`
(optional) Causes source files to be compiled without taking advantage of the built-in version of the IEEE std_logic_1164 package. This typically results in longer simulation times for VHDL programs that use variables and signals of type std_logic.
- `-noaccel <package_name>`
(optional) Turns off acceleration of the specified package in the source code using that package.

`<package_name>` — A VHDL package name.
- `-nocasestaticerror`
(optional) Suppresses case statement static warnings. VHDL standards require that case statement alternative choices be static at compile time. However, some expressions that are globally static are allowed. This switch prevents the compiler from warning on such expressions. If you specify the -pedanticerrors switch, this switch is ignored.
- `-nocheck`
(optional) Disables index checks, range checks, and range checks on floating type values. Disable these checks individually using the -noindexcheck, -norangecheck, and -nofrangecheck arguments, respectively.

Note

If you specify the -nocheck argument and your design has an index or range error, then the tool may show some unexpected behavior.

- `-nocreatelib`
(optional) Stops automatic creation of missing work libraries. Overrides the CreateLib *modelsim.ini* variable. Refer to [CreateLib](#) in the User's Manual.
- `-nodbgsym`
(optional) 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 are source window annotation and textual dataflow.

Specify this switch only if no one using the library will require this information for design analysis purposes.

- **-noDeferSubpgmCheck**
(optional) Causes range and length violations detected within subprograms to be reported as errors (instead of as warnings). An alternative to using this argument is to set the NoDeferSubpgmCheck variable in the *modelsim.ini* file to a value of 1. Refer to [NoDeferSubpgmCheck](#) in the User's Manual.
- **-nofprangecheck**
(optional) Disables range checks on floating type values only.
- **-noFunctionInline**
(optional) Turns off VHDL subprogram inlining for design units that use a local copy of a VHDL package. Useful when the local package has the same name as an MTI supplied package.
- **-noindexcheck**
(optional) Disables index checks to determine whether indexes are within the declared array bounds.
- **-nologo**
(optional) Disables display of the startup banner.
- **-nonstddriverinit**
(optional) Forces ModelSim to match pre-5.7c behavior in initializing drivers in a particular case. 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 did have explicit initial values. This could cause ModelSim to incorrectly use the actual signal's initial value when initializing lower level drivers. This argument does not cause all lower-level drivers to use the actual signal's initial value, only the specific cases where older versions used the actual signal's initial value.
- **-noothersstaticerror**
(optional) Disables warnings caused by array aggregates with multiple choices having "others" clauses that are not locally static. If you specify -pedanticerrors, this switch is ignored.
- **-norangecheck**
(optional) Disables range checks and range checks on floating type values. In some designs, this results in a 2X speed increase. The tool shows the scalar values that are out of range in the following format: ?(N) where N is the current value. To enable the checks, use the -rangecheck argument. By default, range checks are enabled.
- **-note <msg_number> [,<msg_number>, ...]**
(optional) Changes the severity level of the specified message(s) to note. Edit the note variable in the *modelsim.ini* file to set a permanent default. Refer to [note](#) and “[Message Severity Level](#)” in the User's Manual for more information.

<msg_number> — A number identifying the message. Specify multiple message numbers as a comma-separated list.

- -novitalcheck
 - (optional) Disables Vital level 1 and Vital level 0 checks defined in section 4 of the Vital-95 Spec (IEEE Std 1076.4-1995).
- -nowarn <category_number>
 - (optional) Selectively disables a category of warning messages. You can disable all warnings for all compiles through the Main window **Compile > Compile Options** menu command, or the *modelsim.ini* file (refer to [modelsim.ini Variables](#) in the User's Manual).
<category_number> — Specifies one or more numbers corresponding to the categories in [Table 2-6](#). Specify multiple message categories as a comma-separated list.

Table 2-6. 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
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
14	locally static error deferred until simulation run

- -oldconfigvis
 - (optional) Forces vcom to process visibility of VHDL component configurations consistent with prior releases. Default behavior is to comply with Language Reference Manual visibility rules. Refer to vcom -lrmconfigvis, or to the description of the [OldVHDLConfigurationVisibility](#) *modelsim.ini* variable in the User's Manual, for more information.
- -optionset <optionset_name>
 - (optional) Calls an optionset as defined in the *modelsim.ini* file. Refer to the section “[Optionsets](#)” on page 27 for more information.

- **-outf <filename>**
(optional) Specifies a file to save the final list of options to, after recursively expanding all -f, -file and -F files.
- **-pedanticerrors**
(optional) Forces display of an error message (rather than a warning) on a variety of conditions. Refer to “[Enforcing Strict 1076 Compliance](#)” in the User’s Manual for a complete list of these conditions. This argument overrides -nocasestaticerror and -noothersstaticerror.

You can view a complete list of errors by executing the command:

```
verror -kind vcom -pedanticerrors
```

- **-performdefaultbinding**
(optional) Enables default binding that has been disabled through the `RequireConfigForAllDefaultBinding` option in the *modelsim.ini* file. Refer to [RequireConfigForAllDefaultBinding](#) in the User’s Manual.
- **-preserve**
(optional) Forces vcom to preserve the case of letters in object identifiers. You can also preserve letter case with the `PreserveCase` variable in the *modelsim.ini* file. Refer to [PreserveCase](#) in the User’s Manual.
- **-[w]prof=<filename>**
(optional; -prof and -wprof are mutually exclusive) Enables CPU (-prof) or WALL (-wprof) time-based profiling and saves the profile data to <filename>. Customer Support uses output from these arguments for debugging purposes.
- **-proftick=<integer>**
(optional) Sets the time interval between the profile data collections. Default = 10.
- **-quiet**
(optional) Disables output of informative messages about processing the design, such as Loading, Compiling, or Optimizing, but not messages about the design itself.
- **-rangecheck**
(default) Enables range checks. Disable range checks with the -norangecheck argument.
- **-refresh <primary> [<secondary>]**
(optional) Regenerates a library image. 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).
 - <primary> [<secondary>] — Specifies the entity, package, configuration, or module to delete.
 - If <primary> is an entity — Refreshes only that entity, no related architectures.

- If <primary> is a package — Refreshes only the package, and the only legal value of <secondary> is “body”.
- If you specify both <primary> and <secondary> — Updates only the <secondary> architecture, not the entity.

If a dependency checking error occurs which prevents the refresh, use the vcom -force_refresh argument. Refer to the vcom Examples for more information. You can use a specific design name with -refresh to regenerate a library image for that design, but you cannot use a file name.

You cannot specify the <filename> argument when specifying this argument.

- -s
(optional) Instructs the compiler not to load the standard package. Use this argument only if you are compiling the standard package itself.
- -separateConfigLibrary
Allows the declaration of a VHDL configuration to occur in a different library than the entity being configured. Strict conformance to the VHDL standard (LRM) requires that they be in the same library.
- -showsubprograms | -noshowsubprograms
(optional) Toggles viewing of VHDL subprogram scopes between the command line and a GUI window, for example, the Structure window. The default is not to show subprogram scopes.
- -skip abcepx
(optional) Directs the compiler to skip all:
 - a — architectures
 - b — bodies
 - c — configurations
 - e — entities
 - p — packages
 - x — VHDL 2008 context declarations

You can use any combination in any order, but the argument requires at least one option.

- -skipsynthoffregion
(optional) Ignore all constructs within synthesis_off or translate_off pragma regions.
- -smartdbgsym
(optional) Reduces the size of design libraries by minimizing the number of debugging symbol files generated at compile time.

Edit the SmartDbgSym variable in the *modelsim.ini* file to set a permanent default. Refer to [SmartDbgSym](#) in the User’s Manual.

- **-source**
(optional) Displays the associated line of source code before each error message generated during compilation. By default, only the error message displays.

- **-stats [=+ | -]<feature>[,[+ | -]<mode>]**
(optional) Controls display of compiler statistics sent to a logfile, stdout, or the transcript. Specifying -stats without options sets the default features (cmd, msg, and time).

Specify multiple features and modes for each instance of -stats as a comma-separated list. You can specify -stats multiple times on the command line, but only the final instance takes effect.

[+ | -] — Controls activation of the feature or mode. The plus character (+) enables the feature, and the minus character (-) disables the feature. You can also enable a feature or mode by specifying it without the plus (+) character. Setting this switch adds or subtracts features and modes from the settings in the Stats *modelsim.ini* variable.

<feature>

all — Display all statistics features (cmd, msg, perf, time). Mutually exclusive with the none option. Applied first when specified in a string with other options.

cmd — (default) Echo the command line.

msg — (default) Display error and warning summary at the end of command execution.

none — Disable all statistics features. Mutually exclusive with the all option. Applied first when specified in a string with other options.

perf — Display time and memory performance statistics.

time — (default) Display Start, End, and Elapsed times.

Note

 Use -stats=perf+verbose to display information about the operating system and host machine.

<mode>

Modes can be set for a specific feature, or globally for all features. Use the plus (+) or minus (-) character to add or subtract a mode for a specific feature, for example, vcom -stats=cmd+verbose,perf+list. To add or subtract a mode globally for all features, specify the modes in a comma-separated list, for example, vcom -stats=time,perf,list,-verbose. You cannot specify global and feature specific modes together.

kb — Print performance statistics in kilobyte units with no auto-scaling.

list — Display statistics in a Tcl list format when available.

verbose — Display verbose statistics information when available.

Note

Specify vcom -quiet to disable all -stats features.

- **-suppress {<msgNumber> | <msgGroup>} [,<msg_number> | <msgGroup>] ,...]**
(optional) Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message to suppress. You cannot suppress Fatal or Internal messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to [suppress](#) and “[Message Severity Level](#)” in the User’s Manual for more information.

<msgNumber>

A specific message number

<msgGroup>

A value identifying a pre-defined group of messages, based on area of functionality. These groups only suppress notes or warnings. The valid arguments are:

All, GroupNote, GroupWarning, GroupFLI, GroupPLI, GroupSDF, GroupVCD,
GroupVital, GroupWLF, GroupTCHK, GroupPA, GroupLRM

- **-version**
(optional) Returns the version of the compiler as used by the licensing tools.
- **-vitalmemorycheck**
(optional) Enables VITAL level 1 checks.
- **-vmake**
(optional) Generates a complete record of all command line data and files accessed during the compile of a design. The [vmake](#) command uses this data 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 or not they contribute data to the initial compile. This switch can increase compile time in addition to increasing the accuracy of the compile. Refer to the [vmake](#) command for more information.

- **-warning <msg_number>[,<msg_number>,...]**

(optional) Changes the severity level of the specified message(s) to "warning." Edit the warning variable in the *modelsim.ini* file to set a permanent default. Refer to [warning](#) and “[Message Severity Level](#)” in the User’s Manual for more information.

<msg_number> — A number identifying the message. Specify multiple message numbers as a comma-separated list.

- **-warning error**

(optional) Reports all warnings as errors.

-
- **-work <library_name>**

(optional) Maps a library to the logical library work. By default, compiled design units are added to the work library. The specified pathname overrides the pathname specified for work in the project file.

<library_name> — A logical name or pathname of a library.

- **<filename>**

(required, except when you specify -refresh or -force_refresh) Specifies the name of the file containing the VHDL source to compile. Requires at least one filename. You can specify multiple filenames as a space-separated list. You can use wildcards, for example, *.vhd.

If you do not specify a filename, and you are using the GUI, a pop-up dialog box enables you to select options and enter a filename.

Examples

- Compile the VHDL source code contained in the file *example.vhd*.

vcom example.vhd

- Compile unordered files.

vcom -autoorder src1.vhd src4.vhd src3.vhd src2.vhd

The -autoorder argument allows vcom to incrementally build multiple libraries of design units and packages, as long as design units in one library depend on completely compiled design units of another library. For example, if design units of lib1 depend on design units of lib2, and lib2 has no dependences on design units in any other library, the sequence of commands is:

**vcom -work lib2 -autoorder src1.vhd src2.vhd
vcom -work lib1 -autoorder src3.vhd src4.vhd**

Use -autoorder to compile and combine libraries. To compile a design into a library, first compile all dependent or resource libraries in an ordered fashion without using the -autoorder argument. Next, compile the design into one library using the -autoorder argument.

**vcom -work lib1 src1.vhd src2.vhd
vcom -work lib2 src3.vhd src4.vhd
vcom -autoorder designsrc/*.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**

- 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, meaning that, 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. VHDL users typically expect that the explicit declaration can to hide the implicit declaration.

ARITHMETIC."=" (left, right)

- The -work argument specifies mylib as the library to regenerate. The -refresh argument 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

- Enable the display of Start, End, and Elapsed time, and a message count summary. Disables echoing of the command line.

vcom -stats=time,-cmd,msg

- The first -stats option is ignored. The none option disables all *modelsim.ini* settings and then enables the perf option.

vcom -stats=time,cmd,msg -stats=none,perf

vdel

Deletes a design unit from a specified library.

Syntax

```
vdel [-lib <library_path>] [-modelsimini <path/modelsim.ini>] [-verbose] {-all | <primary> <arch_name>} | -obj {<object_info>} | -dpiobj [<object_info>]
```

Arguments

- **-all**
(optional) Deletes an entire library.

Caution



You cannot recover deleted libraries. You are not prompted for confirmation.

- **-dpiobj [<object_info>]**
(optional) Deletes auto-compiled DPI object files.
 <object_info> — Specifies the type of object files to remove, as reported in the output of the vdir -obj command. The object file types are:
 <compiler> — a string identifying the compiler, such as gcc-3.3.1.
 <platform> — a string identifying the platform.
 <platform-compiler> — a string identifying a compiler/platform pair, such as linux_gcc-3.2.3.
 all — Specifies to remove all object files reported in the output of the vdir -obj command.
- **-lib <library_path>**
(optional) Specifies the location of the library containing the design unit to delete. The default is to delete the design unit from the work library.
 <library_path> — The logical name or pathname of the library.
- **-modelsimini <path/modelsim.ini>**
(optional) Loads an alternate initialization file to replace the current initialization file. Overrides the file path specified in the MODELSIM environment variable.
 <path/modelsim.ini> — Specifies either an absolute or relative path to the initialization file, including the file name. On Windows systems, the path separator is a forward slash (/).
- **-obj {<object_info>}**
(optional) removes directories containing DPI object files.

<object_info> — Specifies the type of directory to remove, as reported in the output of the vdir -obj command. The directory types are:

<compiler> — a string identifying the compiler, such as gcc-3.3.1.

<platform> — a string identifying the platform.

<platform-compiler> — a string identifying a compiler/platform pair, such as linux_gcc-3.2.3.

all — Specifies to remove all directories reported in the output of the vdir -obj command.

- <primary> [<arch_name>]

(required unless -all is used) Specifies the entity, package, configuration, or module to delete.

<arch_name> — Specifies the name of an architecture to delete. If omitted, all architectures for the specified entity are deleted. Invalid for a configuration or a package.

- -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

Lists the contents of a design library and checks the compatibility of a vendor library.

Syntax

```
vdir [-l | [-prop <prop>]] [-r] [-all | [-lib <library_name>]] [<design_unit>] [-modelsimini  
      <path/modelsim.ini>]
```

Description

If vdir cannot read a vendor-supplied library, the library may not be compatible with ModelSim.

Arguments

- **-all**
(optional) Lists the contents of all libraries included in the Library section of the active *modelsim.ini* file. Refer to [modelsim.ini Variables](#) in the User's Manual for more information.
- **<design_unit>**
(optional) Specifies the design unit to search for in the specified library. If the design unit is a VHDL entity, its architectures are listed. The default is to list all entities, configurations, modules, packages, and optimized design units in the specified library.
- **-l**
(optional) Prints the version of vcom/vlog used to compile each design unit, and any compilation options. Also prints the object-code version number that indicates which versions of the executable and ModelSim are compatible.
- **-lib <library_name>**
(optional) Specifies the logical name or the pathname of a library to list. The default is to list the contents of the work library.
 <library_name> — The logical name or pathname of a library.
- **-modelsimini <path/modelsim.ini>**
(optional) Loads an alternate initialization file to replace the current initialization file. Overrides the file path specified in the MODELSIM environment variable.
 <path/modelsim.ini> — Specifies either an absolute or relative path to the initialization file, including the file name. On Windows systems, the path separator is a forward slash (/).
- **-prop <prop>**
(optional) Reports on a specified design unit property.
 <prop> — Specifies a Design Unit Property, as listed in [Table 2-7](#). If you do not specify a value for <prop>, an error message displays.

Table 2-7. Design Unit Properties

Value of <prop>	Description
archcfg	configuration for arch
body	needs a body
cmpltime	compilation time
default	default options
dir	source directory
dpnd	depends on
entcfg	configuration for entity
fulloptions	Full compile options
inline	module inlined
lock	lock/unlock status
lrm	language standard
mtime	source modified time
name	short name
opcode	opcode format
options	compile options
pdu	preoptimized design unit
root	optimized Verilog design root
src	source file
top	top level model
ver	version string
vlogv	Verilog version

- -r

(optional) Prints architecture information for each entity in the output.

Examples

- List the architectures associated with the module named and2 that reside in the default library work.

```
vdir -I and2
```

```
# Library vendor : Model Technology
# Maximum unnamed designs : 3
# MODULE and2
#     Verilog version: <XO@d;_mSdz@12Fz9b]_Z3
#     Version string: 3EdggZ>V3z51fE;>K[51?2
#     Source directory: C:\examples\dataflow_verilog
#     Source modified time: Tue Apr 28 22:48:56 2009
#     HDL source file: gates.v
#     Source file: gates.v
#     Start location: gates.v:18
#     Opcode format: 10.1a; VLOG SE Object version 51
#     Optimized Verilog design root: 1
#     VHDL language standard: 1
#     Compile options: -L mtiAvm -L mtiOvm -L mtiUvm -L mtiUPF
#     Debug Symbol file exists
```

vencrypt

Encrypts Verilog and SystemVerilog code.

Syntax

```
vencrypt <filename> [<filename>...] -common <filename>] [-data_method=[aes128 | aes192 |  
aes256]] [-d <dirname>] [-e <extension>] [-f <cmdfile>] [-hea <headerfile>] [-keyring  
<directory>] [-logfile <logfile> | -l <logfile>] [-o <outputfile>] [-p <prefix>] [-quiet] [[-stats  
[=+ | -]<feature>[,[+ | -]<mode>]]] [-toolblock <keyfile>[,<rightsfile>]] [-wholefile]
```

Note

 The arguments shown are only for the automatic encryption of Verilog files. SystemVerilog is not supported.

```
vencrypt [-autotprotect] [-autoprotect_all] [-auto2protect] [-auto3protect] [-auto5protect] [-  
pli_unprotected]
```

Description

The vencrypt command does not pre-process code before encryption, so macros and other `directives are unchanged, allowing you to deliver encrypted IP with undefined macros and `directives.

Upon execution of this command, the filename extension changes 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 entire file using the following default encryption key:

```
`pragma protect version = 1  
`pragma protect key_keyowner = "Mentor Graphics Corporation"  
`pragma protect key_keyname = "MGC-VERIF-SIM-RSA-2"  
`pragma protect key_method = "rsa"  
`pragma protect key_block encoding = ( enctype = "base64" )  
`pragma protect data_method = "aes128-cbc"
```

You can use the -wholefile argument to force the command to encrypt an entire file. This switch ignores existing pragmas within the input file. If you specify the -data_method argument on the command line, the command uses it rather than the **data_method** pragma defined in the code above (aes128-cbc).

If you specify the -allow_veloce argument with no directives in the input files, then the command adds the following:

```
`pragma protect key_keyowner = "Mentor Graphics Corporation"  
`pragma protect key_keyname = "MGC-VELOCE-RSA"  
`pragma protect key_method = "rsa"  
`pragma protect key_block encoding = ( enctype = "base64" )
```

The vencrypt command contains a number of arguments for automatic encryption of Verilog only files without the use of pragmas and which only support IEEE 1735 version 0. Note the following:

- Automatic encryption does not require any embedded protect pragmas nor the specification of any encryption key.
- In autoprotect mode, the command ignores any embedded protect directives.
- Encryption is based on modules.
- The -pli_unprotected argument enables PLI users to access encrypted modules.

You can use only one of the "autoprotect" arguments at a time. You cannot use autoprotect mode with the following vencrypt arguments:

- allow_venge
- wholefile
- data_method
- common
- toolblock
- keyring
- header

Arguments

- <filename> [<filename> . . .]
(required) Specifies the name of the Verilog source code file to encrypt. Requires one filename. You can enter multiple filenames as a space-separated list. You can use wildcards. Default encryption pragmas are used, as described above, if no encryption directives are found during processing.
- -autotprotect
Encrypts the text that appears between a Verilog module <name> and endmodule statements. The module names remain user visible. Encrypts the parameter and port lists. Does not encrypt any text that is not contained within a Verilog module.
- -autoprotect_all
Encrypts the entire file in one block of encrypted code, fully protecting every design unit, including code outside any Verilog module.
- -auto2protect
Encrypts an entire module, except for the port list. The parameter list is encrypted. (Does not encrypt any `define and `include statements inside the module.)

- **-auto3protect**

Encrypts an entire Verilog module, except for the parameter and port list. (Does not encrypt `define and `include statements inside the module.)

- **-auto5protect**

Encrypts an entire Verilog module, except for the parameter and port list. Also encrypts content between `ifdef and `ifndef construct directives, so that only the enclosing `ifdef/`endif are visible. Does not encrypt the meta comments between //unprotect and //endunprotect.

- **-pli_unprotected**

Used only in conjunction with the -auto2protect, -auto3protect, and -auto5protect arguments. The resulting encrypted files follow the results of the given auto*protect option, but PLI users can access encrypted data in the -pli_unprotected modules. PLI users cannot access modules encrypted without this option.

- **-common <filename>**

Specifies a file containing information to make common to all the tool blocks. Common information includes any directives you intend for the common block. You can also include **author**, **author_info**, and **data_method** directives. The file can contain **begin_commonblock** and **end_commonblock** directives, but these are ignored by the command. The file can also contain single-line comments ("// for Verilog). The command searches the specified file for the location of the "keyring" directory. You can specify only one -common file. If you specify only one file, then you must supply at least one -toolblock argument.

- **-d <dirname>**

Specifies where to save encrypted Verilog files. If you do not specify a directory, the current working directory is used.

<dirname> — Specifies the directory to contain the encrypted Verilog or SystemVerilog files. The original file extension (.v for Verilog and .sv for SystemVerilog) is preserved.

- **-data_method=[aes128 | aes192 | aes256]**

Specifies the symmetric encryption method to use with the random session key when encrypting text in a data_block. The data_block consists of the characters that appear between the begin and end directives in the input file. If you do not specify a common or -toolblock argument, then the command creates an IEEE 1735 version 1 compliant encryption envelope using the Mentor Graphics public encryption key. This command line setting overrides any -data_method directives appearing in the input file.

aes128 — AES key size of 128. This is the default.

aes192 — AES key size of 192.

aes256 — AES key size of 256.

- **-e <extension>**
(optional) Specifies a filename extension.
 <extension> — Any alpha-numeric string.
- **-f <cmdfile>**
(optional) Specifies a file containing command line arguments. Allows you to reuse complex arguments without retyping. Allows nesting of -f options.
 <cmdfile> — Specifies the name of a file containing command line arguments.
- **-hea <headerfile>**
(optional) Concatenates header information into all design files listed with <filename>. Enables you to easily add the same `pragma protect information to a large number of files. Allows you to pass files to the vencrypt utility that do not contain the `pragma protect information about how to encrypt the file.
 <headerfile> — Specifies an existing file.
- **-keyring <directory>**
Specifies the directory containing the common and toolblock files. If you do not specify the directory, the command assumes the directory is underneath the top of the installation directory <top_of_installation_dir>/keyring. This option is useful only if using the -toolblock argument and -common.
- **-logfile <logfile> | -l <logfile>**
(optional) Redirects log output to the file designated by <logfile>.
 <logfile> — Specifies a file for saving output.
- **-o <outputfile>**
(optional) Combines all encrypted output into a single file.
 <outputfile> — 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.
- **-stats [=+ | -]<feature>[,[+ | -]<mode>]**
(optional) Controls display of statistics sent to a logfile, stdout, or the transcript. Specifying -stats without options sets the default features (cmd and msg).
Specify multiple features and modes for each instance of -stats as a comma-separated list. You can specify -stats multiple times on the command line, but only the final instance takes effect.

[+ | -] — Controls activation of the feature or mode, where the plus character (+) enables the feature and the minus character (-) disables the feature. You can also enable a feature or mode by specifying it without the plus (+) character. Setting this switch adds or subtracts features and modes from the default settings "cmd,msg".

Features

all — Display all statistics features (cmd, msg, perf). Mutually exclusive with none option. When specified in a string with other options, all is applied first.

cmd — (default) Echo the command line.

msg — (default) Display error and warning summary at the end of command execution.

none — Disable all statistics features. Mutually exclusive with all option. When specified in a string with other options, none is applied first.

perf — Display time and memory performance statistics.

time — Has no effect and is ignored.

Modes

Modes can be set for a specific feature, or set globally for all features. To add or subtract a mode for a specific feature, specify using the plus (+) or minus (-) character with the feature, for example, vencrypt -stats=cmd+verbose,perf+list. To add or subtract a mode globally for all features, specify the modes in a comma-separated list, for example, vencrypt -stats=time,perf,list,-verbose. You cannot specify global and feature specific modes together.

kb — Print performance statistics in kilobyte units with no auto-scaling.

list — Display statistics in a Tcl list format when available.

verbose — Display verbose statistics information when available.

Note



vencrypt -quiet disables all default or user-specified -stats features.

- -toolblock <keyfile>[,<rightsfile>]

Specifies the root name of a file that contains directives intended for a tool block.

keyfile — The root name of the file. Generally, it is the name of a key-key (key_keyname) contained within the file. This file is in the keyring directory, and must include either a ".deprecated" or ".active" suffix to the base root name. The command does not use the suffix as part of the specified filename.

rightsfile — Optionally, a "rights file" name may appear after the keyfile name; This file contains control directives for this particular tool block. Control directives are used to define rights. The rights file must have a ".rights" suffix.

- -wholefile

This switch ignores any `pragma protect directives that appear in the input file. The command encrypts all the specified files, referencing the -common and -toolbox arguments

to determine how to encrypt the files. If you do not specify the -common or -toolblock arguments, then the command creates a version 1 (V1) encryption envelope, with only one key block: the default key for Mentor Graphics Corporation.

Examples

- 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.

- Enable the display of message count summary. Echoing of the command line is disabled.

```
vencrypt -stats=msg,-cmd
```

- The first -stats option is ignored. The none option disables all default settings and then enables the perf option.

```
vencrypt -stats=msg,cmd -stats=none,perf
```

Related Topics

[vhencrypt](#)

verror

Returns a detailed description about a message number or a list of messages related to a specified portion of the product.

Syntax

```
verror [-fmt | -full] <msgNum> [,]...
verror [-fmt | -full] [-kind <tool>] -all
verror [-kind <tool>] {-pedanticerrors | -permissive | -suppressibleerrors}
```

Arguments

- **-fmt | -full**

(optional) Specifies the type and amount of information to return.

-fmt

Returns the format string used in the message.

-full

Returns the format string and complete text associated with the message.

- **[-kind <tool>] -all**

(required when not specifying <msgNum>) Returns information about all messages associated with a specified tool, where <tool> can be one of the following:

aid	hm_entity	mc2com	qverilog
sccom	scgenmod	sdfcomp	sm_entity
vcd2wlf	vcom	vcovkill	vdel
vdir	vencrypt	vgencomp	vish
vlib	vlog	vmake	vmap
vopt	vsim	wlf	wlf2log
wlfman	wlfrecover		

- **[-kind <tool>] {-pedanticerrors | -permissive | -suppressibleerrors}**

(optional) Specifies filtering for messages according to the message type.

<tool>

Any of the values allowed for the -kind argument.

-pedanticerrors

Display messages that are reported as errors due to adhering to a more strict interpretation of the LRM.

-permissive

Display messages reported as warnings that would be displayed as errors if you use `vsim -pedanticerrors`.

-suppressibleerrors

Display messages that you can suppress from the command line or *modelsim.ini* file.

- **<msgNum>**

(required when not specifying `-all`) 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 use either a comma-separated or a space-separated list to specify `<msgNum>` any number of times for one `verror` command.

Optionally, you can specify the toolname prior to the message number, in the same way it appears in an error message. For example:

```
verror vsim-5003
```

Examples

- 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 can enter:

```
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

Takes a compiled Verilog module from a library and writes its equivalent VHDL component declaration to standard output.

Syntax

```
vgencomp [-lib <library_name>] [-b] [-bool] [-modelsimini <path/modelsim.ini>] [-s] [-v] [-work <name>] <module_name>
```

Description

Optional switches allow you to generate bit or vl_logic port types. Default is to generate std_logic port types.

Arguments

- **-lib <library_name>**
(optional) Specifies the working library. 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.
- **-bool**
(optional) Causes vgencomp to generate boolean port types.
- **-modelsimini <path/modelsim.ini>**
(optional) Loads an alternate initialization file to replace the current initialization file. Overrides the file path specified in the MODELSIM environment variable.
<path/modelsim.ini> — Specifies either an absolute or relative path to the initialization file, including the file name. On Windows systems the path separator is 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.
- **-work <name>**
(optional) Specifies the name of the work library. Default is the library containing the module.
- **<module_name>**
(required) Specifies the name of the Verilog module to access.

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

Encrypts VHDL code contained within encryption envelopes.

Syntax

```
vhencrypt <filename> [<filename> . . .] [-common <filename>][-d <dirname>] [-common  
    <filename>[-data_method=[aes128 | aes192 | aes256] [-e <extension>] [-f <cmdfile>] [-hea  
    <headerfile>] [-keyring <directory>] [-logfile <logfile> | -l <logfile>] [-o <outputfile>]  
    [-p <prefix>] [-quiet] [-stats [=+ | -]<feature>[,[+ | -]<mode>]] [-toolblock  
    <keyfile>[,<rightsfile>][ [-wholefile]
```

Description

Upon execution, vhencrypt changes the *.vhd* filename extension to *.vhdp*, and the *.vhdl* filename extension to *.vhdlp*.

If you do not specify any encryption pragmas or controls, the vhencrypt command uses by default information in the keyring directory.

Since the code is not compiled before encryption, dependent packages and design units do not have to exist before encryption. You must follow the vhencrypt command with a compile command – such as vcom – for the design to be compiled.

Arguments

- <filename> [<filename> . . .]
(required) Specifies the name of the VHDL source code file to encrypt. One filename is required. Enter multiple filenames as a space-separated list. Allows wildcards.
- -common <filename>
Specifies a file containing information to make common to all the tool blocks. Common information includes any directives you intend for the common block. You can also include author, author_info, and data_method directives. The file can contain begin_commonblock and end_commonblock directives, but these are ignored by the command. The file can also contain single-line comments ("// " for Verilog). The command searches the specified file for the location of the "keyring" directory. You can specify only one -common file. If you specify only one file, then you must supply at least one -toolblock argument.
- -d <dirname>
(optional) Specifies where to save encrypted VHDL files. Defaults to the current working directory, if you do not specify a directory.
 - <dirname> — Specifies the directory to contain the encrypted VHDL files. Preserves the original file extension (*.vhd* or *.vhdl*).
- -data_method=[aes128 | aes192 | aes256]
Specifies the symmetric encryption method to use with the random session key when encrypting text in a data_block. The data_block consists of the characters that appear

between the begin and end directives in the input file. If you do not specify a common or -toolblock argument, then the command creates an IEEE 1735 version 1 compliant encryption envelope using the Mentor Graphics public encryption key. This command line setting overrides any -data_method directives appearing in the input file.

 aes128 — AES key size of 128. This is the default.

 ase192 — AES key size of 192.

 aes256 — AES key size of 256.

- **-e <extension>**

(optional) Specifies a filename extension to apply to the encrypted file.

 <extension> — Any alpha-numeric string.

- **-f <cmdfile>**

(optional) Specifies a file with more command line arguments. Allows reuse of complex arguments without retyping. Allows nesting of -f options.

 <cmdfile> — Specifies the name of a file containing command line arguments.

- **-hea <headerfile>**

(optional) Concatenates header information into all design files listed with <filename>. Enables you to pass vhencrypt a large number of files that do not contain the encryption information in the form of the `protect compiler directives. Enables you to avoid having to edit hundreds of files to add the same encryption information.

 <headerfile> — Specifies an existing file.

- **-keyring <directory>**

Specifies the directory containing the common and toolblock files. If you do not specify the directory, the command assumes the directory is underneath the top of the installation directory <installation>/keyring. This option is useful only if using the -toolblock argument and -common.

- **-logfile <logfile> | -l <logfile>**

(optional) Redirects log output to the file designated by <logfile>.

 <logfile> — Specifies a file for saving output.

- **-o <outputfile>**

(optional) Combines all encrypted output into a single file.

 <outputfile> — 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.

- **-stats [= [+ | -]<feature>[,[+ | -]<mode>]**

(optional) Controls display of statistics sent to a logfile, stdout, or the transcript. Specifying -stats without options sets the default features (cmd and msg).

Specify multiple features and modes for each instance of -stats as a comma-separated list. You can specify -stats multiple times on the command line, but only the final instance takes effect.

[+ | -] — Controls activation of the feature or mode, where the plus character (+) enables the feature, and the minus character (-) disables the feature. You can also enable a feature or mode by specifying it without the plus (+) character. Setting this switch adds or subtracts features and modes from the default settings "cmd,msg".

Features

all — Display all statistics features (cmd, msg, perf). Mutually exclusive with the none option. When specified in a string with other options, all is applied first.

cmd — (default) Echo the command line.

msg — (default) Display error and warning summary at the end of command execution.

none — Disable all statistics features. Mutually exclusive with the all option. When specified in a string with other options, none is applied first.

perf — Display time and memory performance statistics.

time — Has no effect and is ignored.

Modes

Modes can be set for a specific feature, or set globally for all features. To add or subtract a mode for a specific feature, use the plus (+) or minus (-) character with the feature, for example, vhencrypt -stats=cmd+verbose,perf+list. To add or subtract a mode globally for all features, specify the modes in a comma-separated list, for example, vhencrypt -stats=time,perf,list,-verbose. You cannot specify global and feature specific modes together.

kb — Print performance statistics in kilobyte units with no auto-scaling.

list — Display statistics in a Tcl list format when available.

verbose — Display verbose statistics information when available.

Note

 You can use vhencrypt -quiet to disable all default or user-specified -stats features.

- **-toolblock <keyfile>[,<rightsfile>]**

Specifies the root name of a file that contains directives intended for a tool block.

keyfile — The root name of the file. Generally, it is the name of a key-key (key_keyname) contained within the file. This file is in the keyring directory, and must include either a ".deprecated" or ".active" suffix to the base root name. The command does not use the suffix as part of the specified filename.

rightsfile — Optionally, a "rights file" name may appear after the keyfile name; This file contains control directives for this particular tool block. Control directives are used to define rights. The rights file must have a ".rights" suffix.

- **-wholefile**

This switch ignores any `pragma protect directives that appear in the input file. The command encrypts all the specified files, referencing the -common and -toolbox arguments to determine how to encrypt the files. If you do not specify the -common or -toolbox arguments, then the command creates a version 1 (V1) encryption envelope, with only one key block: the default key for Mentor Graphics Corporation.

Examples

- Enable the display of message count summary. Echoing of the command line is disabled.

vhencrypt -stats=msg,-cmd,

- The first -stats option is ignored. The none option disables all default settings and then enables the perf option.

vhencrypt -stats=msg,cmd -stats=none,perf

Related Topics

[vencrypt](#)

view

Opens the specified window or, if specified without arguments, returns a list of all open windows in the current layout.

Syntax

```
view <window_type>...[-aliases][-names] [-title {New Window Title}]  
      [-undock {[ -icon] [-height <n>] [-width <n>] [-x <n>] [-y <n>]}] | -dock]
```

Description

To remove a window, use the [noview](#) command.

If you do not specify a window name, any view command options apply to the currently open windows. Refer to examples for additional details.

Arguments

- <window_type>...

(required) Specifies the window type to view. You do not need to enter the full type name (see the examples below); accepts implicit wildcards; accepts multiple window types.

Available window types are:

assertions	atv	browse	calltree
canalysis	capacity	classgraph	classtree
coveragegroups	dataflow	details	duration
exclusions	fcovers	files	fsmlist
fsmview	instance	library	list
locals	memdata	memory	msgviewer
objects	process	profiledetails	project
ranked	runmgr	schematic	source
stackview	structural	structure	tracker
transaction	transcript	uvmdetails	watches

wave

Not all windows are available with all variants of ModelSim and Questa SIM

- **-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.
- **-title {New Window Title}**
(optional) Specifies the window title of the designated window.
 {New Window Title} — Any string. You must enclose strings that contain spaces in curly braces or double quotes (" "), for example, "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

virtual count

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.

Accepts unique abbreviations.

- `-unsaved`

(optional) Reports the count of only those virtuals that have not been saved to a macro file.

Related Topics

[virtual define](#)

[virtual save](#)

[virtual show](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual define

Prints the definition of the virtual signals, functions, or regions to the transcript, in the form of a command that can be used to re-create the object.

Syntax

```
virtual define [-kind {implicits | explicits}] <pathname>
```

Arguments

- `-kind {implicits | explicits}`

(optional) Prints only a subset of virtuals to the transcript.

`implicits` — Virtual signals created internally by the tool.

`explicits` — Virtual signals explicitly created by a user, such as with the `virtual signal` command.

Accepts unique abbreviations.

- `<pathname>`

(required) Specifies the path to the virtual(s) for which you want definitions. Allows wildcards.

Examples

- Show the definitions of all the virtuals you have explicitly created.

```
virtual define -kind explicits *
```

Related Topics

[virtual describe](#)

[virtual show](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual delete

Removes the matching virtuals.

Syntax

`virtual delete [-kind {implicits | explicits}] <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.

Accepts unique abbreviations.

- `<pathname>`

(required) Specifies the path to the virtual(s) to delete. Allows wildcards.

Examples

- Delete all of the virtuals you have explicitly created.

`virtual delete -kind explicits *`

Related Topics

[virtual signal](#)

[virtual function](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual describe

Prints a complete description of the data type of one or more virtual signals to the transcript.
Similar to the describe command.

Syntax

```
virtual describe [-kind {implicits | explicits}] <pathname>
```

Arguments

- `-kind {implicits | explicits}`

(optional) Prints only a subset of virtuals to the transcript.

`implicits` — virtual signals created internally by the product.

`explicits` — virtual signals explicitly created by a user, such as with the `virtual signal` command.

Accepts unique abbreviations.

- `<pathname>`

(required) Specifies the path to the virtual(s) for which you want descriptions. Allows wildcards.

Examples

- Describe the data type of all virtuals you have explicitly created.

```
virtual describe -kind explicits *
```

Related Topics

[virtual define](#)

[virtual show](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual expand

Prints a list of all of the non-virtual objects contained in the specified virtual signal(s) to the transcript.

Syntax

`virtual expand [-base] <pathname> ...`

Arguments

- `-base`

(optional) Outputs the root signal parent in place of a subelement. For example, the result of the following command:

```
vcd add [virtual expand -base myVirtualSignal]
```

is:

```
vcd add signala signalb signalc
```

- `<pathname>`

(required) Specifies the path to the signals and virtual signals to expand. Allows wildcards, and you can specify any number of paths.

Description

The virtual expand command enables you to create a list of arguments for a command that does not accept or understand virtual signals.

Examples

- Add the elements of a virtual signal to the VCD file.

In the Tcl language, the square brackets specify to execute the enclosed command first ("virtual expand ..."), then substitute the result into the surrounding command.

```
vcd add [virtual expand myVirtualSignal]
```

If myVirtualSignal is a concatenation of signal_a, signal_b.rec1 and signal_c(5 downto 3), the resulting command after substitution is:

```
vcd add signala signalb.rec1 {signalc(5 downto 3)}
```

The slice of signal_c is enclosed in curly braces, because it contains spaces.

Related Topics

[virtual signal](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual function

Creates a new signal, used by the GUI (not the kernel), that consists of logical operations on existing signals and simulation time.

Syntax

```
virtual function [-env <path>] [-install <path>] [-delay <time> <unit>] {<expressionString>}  
    <name>
```

Description

The virtual function command cannot handle bit selects and slices of Verilog registers. Refer to “[Syntax and Conventions](#)” on page 13 for more details on syntax.

If a virtual function references more than a single scalar signal, it displays as an expandable object in the Wave and Objects windows. The children correspond to the inputs of the virtual function. You can expand the function in the Wave window to compare the values of the input waveforms.

You can also use virtual functions to gate the List window display.

Note

 The virtual function and virtual signal commands are interchangeable. The product keeps track of whether you have created a signal or a function with the commands, and maintains 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.

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 do not all have to be full paths.
 <path> — Specifies a relative path to the signal(s).
- **-install <path>**
(optional) Causes the newly-created signal to become a child of the specified region. If you do not specify -install, 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 is automatically placed in region virtuals:/ Functions.
 <path> — Specifies a relative path to the signal(s). On Windows systems the path separator is a forward slash (/).

- **-delay <time> <unit>**

(optional) Specifies a value by which to delay the virtual function. You can use negative values to look forward in time. Refer to the examples below for more details.

<time> — Specified as an integer or decimal number. Defaults to the current simulation units, if you do not specify <unit>.

<unit> — (optional) Specifies a unit of time. Valid VHDL time units are: fs, ps, ns, us, ms, sec, min, and hr. You must enclose <time> and <unit> within curly braces ({}).

- {<expressionString>}

(required) A text string expression, enclosed in curly braces ({}) using the “[GUI_expression_format](#)” on page 33.

- <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 you use VHDL extended identifier notation, <name> needs to be enclosed in double quotes (" ") or 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 install 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] == 0xfb } 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. You can add the virtual function siga_diff 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 does not 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

<code>add log</code> and <code>log</code>	<code>delete</code>	<code>describe</code>
<code>examine</code>	<code>find</code>	<code>restart</code>
<code>searchlog</code>	<code>show</code>	

Commands not compatible with virtual functions

<code>drivers</code>	<code>force</code>	<code>noforce</code>
<code>vcd add</code>	<code>when</code>	

Related Topics

[virtual count](#)
[virtual define](#)
[virtual delete](#)
[virtual describe](#)
[virtual expand](#)
[virtual hide](#)
[virtual log](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

[virtual nohide](#)
[virtual nolog](#)
[virtual region](#)
[virtual save](#)
[virtual show](#)
[virtual signal](#)
[virtual type](#)

virtual hide

Hides the specified real or virtual signals from display in the Objects window.

Syntax

```
virtual hide {{[-kind {implicits | explicits}] | [-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.

Accepts unique abbreviations.

- `-region <path>`

(optional) Specifies a region of design space in which to look for the signal names.

`<path>` — Specifies an absolute or relative path to the signal(s). On Windows systems, the path separator is a forward slash (/).

- `<pattern>`

(required) Indicates which signal names to use to find the signals to hide. Allows wildcards, and you can specify any number of names or patterns.

Description

You can use the `virtual hide` command to replace an expanded bus with a user-defined bus. Use the `virtual nohide` command to make the signals reappear.

Related Topics

[virtual nohide](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual log

Causes the kernel to log the simulation-mode dependent signals of the specified virtual signals.

Note

 When you use wildcard patterns to specify signals to log, normal signals that match the patterns are also logged, unless you include the -only option. You can unlog the signals with the virtual nolog command.

Syntax

```
virtual log {[ -kind {implicits | explicits} ] | [-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.
- Accepts unique abbreviations.
- **-region <path>**
(optional) Specifies a region of design space in which to look for signals to log.

<path> — Specifies an absolute or relative path to the signal(s). On Windows systems the path separator is a forward slash (/).
- **-recursive**
(optional) Specifies that the scope of the search is to descend recursively into subregions. Omitting -recursive limits the search to the selected region.
- **-only**
(optional) Specifies to log only virtual signals (as opposed to all signals) found by a *<pattern>* containing a wildcard.
- **-in**
(optional) Specifies that the kernel is to log data for ports of mode IN whose names match the specification.
- **-out**
(optional) Specifies that the kernel is to log data for ports of mode OUT whose names match the specification.

- **-inout**
(optional) Specifies that the kernel is to log data for ports of mode INOUT whose names match the specification.
- **-internal**
(optional) Specifies that the kernel is to log data for internal (non-port) objects whose names match the specification.
- **-ports**
(optional) Specifies that the kernel is to log data for all ports.
- **<pattern>**
(required) Specifies the signal names or wildcard patterns to use to find the signals to log.
You can specify any number of names or wildcard patterns.

Related Topics

[Virtual Objects \[ModelSim User's Manual\]](#)

[virtual nolog](#)

virtual nohide

Reverses the effect of a virtual hide command, causing the specified real or virtual signals to reappear the Objects window.

Syntax

```
virtual nohide {[-kind {implicits | explicits}] | [-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.

Accepts unique abbreviations.

- [-region <path>](#)

(optional) Specifies a region of design space in which to look for the signal names.

 <path> — Specifies an absolute or relative path to the signal(s). On Windows systems the path separator is a forward slash (/).

- <pattern>

(required) Indicates which signal names or wildcard patterns to use in finding the signals to hide. Allows wildcards, and you can specify any number of names or patterns.

Related Topics

[virtual hide](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual nolog

Reverses the effect of a virtual log command, causing the simulation-dependent signals of the specified virtual signals to be excluded ("unlogged") by the kernel.

Note

 If you use wildcard patterns to specify signals, the command also unlogs any normal signals it finds, unless you include the -only option.

Syntax

```
virtual nolog {[ -kind {implicits | explicits} ] | [-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.
- Accepts unique abbreviations.
- **-region <path>**
(optional) Specifies a region of design space in which to look for signals to unlog.
<path> — Specifies an absolute or relative path to the signal(s). On Windows systems the path separator is a forward slash (/).
- **-recursive**
(optional) Specifies that the scope of the search is to descend recursively into subregions. Omitting -recursive limits the search to the selected region.
- **-only**
(optional) Specifies to unlog only virtual signals (as opposed to all signals) found by a <pattern> containing a wildcard .
- **-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 the signal names or wildcard patterns to use in finding the signals to unlog. |Allows wildcards, and you can specify any number of names or patterns.

Related Topics

[virtual log](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual region

Creates a new user-defined design hierarchy region.

Note

 You cannot use virtual regions in the [when](#) command.

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 for the new region.

Related Topics

[virtual function](#)

[virtual signal](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual save

Saves the definitions of virtuals to a file named virtual.do in the current directory.

Syntax

`virtual save [-kind {implicits | explicits}] [-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.

Accepts unique abbreviations.

- `-append`

(optional) Specifies to save only virtuals that are not already saved or were not read in from a macro file. Appends these unsaved virtuals to the specified or default file.

- `<filename>`

(optional) The name of the file containing the definitions. If you do not specify `<filename>`, the default virtual filename (virtuals.do) is used. You can specify a different default in the `pref.tcl` file.

Related Topics

[virtual count](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual show

Lists the full path names of all explicitly defined virtuals.

Syntax

`virtual show [-kind {implicits | explicits}]`

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.

Accepts unique abbreviations.

Related Topics

[virtual define](#)

[virtual describe](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual signal

Creates a new signal, known only by the GUI (not the kernel), that consists of concatenations of the signals and sub-elements specified in <expressionString>.

Syntax

```
virtual signal [-env <path>] [-install <path>] [-delay <time> <unit>] {<expressionString>}  
          <name>
```

Description

The virtual signal command cannot handle bit selects and slices of Verilog registers. Please see “Concatenation of Signals or Subelements” on page 40 for more details on syntax.

Note

 The virtual function and virtual signal commands are interchangeable. The GUI keeps track of whether you have created a signal or a function with the commands, and maintains 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.

Arguments

- **-env <path>**
(optional) Specifies a hierarchical context in <expressionString> so that you do not have to enter a full path for each signal name.

 <path> — Specifies a relative path to the signal(s). On Windows systems the path separator is a forward slash (/).
- **-install <path>**
(optional) Causes the newly-created signal to become a child of the specified region. If you do not specify -install, 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 is automatically placed in region virtuals:/Signals.

 <path> — Specifies a relative path to the signal(s). On Windows systems the path separator is a forward slash (/).
- **-delay <time> <unit>**
(optional) Specifies the value by which the virtual function is delayed. You can use negative values to look forward in time. Refer to the examples below for more details.

 <time> — Specified as an integer or decimal number. If you do not specify <unit>, the default is the current simulation units.

 <unit> — (optional) A suffix specifying a unit of time. 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 ({}).

- {<expressionString>}
(required) A text string expression, enclosed in curly braces ({}) using the “[GUI_expression_format](#)” on page 33.
- <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 enclosed in double quotes ("") or 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 (for example, num28, num27, and so on) 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 of Signals or Subelements” on page 40 for further details.

```
virtual signal {(concat_reverse)(bus1 & bus2[7:4])} newbus
```

Commands fully compatible with virtual signals

add	add	add
list	log or	wave
	log	
delet	descr	exam
e	ibe	ine
find	force	restar
	and	t
	nofor	
	ce	
searc	show	
hlog		

Commands compatible with virtual signals using [virtual expand <signal>]

driv	vcd add
ers	

Commands not currently compatible with virtual signals

- wh
- e
- n

Related Topics

[virtual count](#)
[virtual describe](#)
[virtual log](#)
[virtual region](#)
[virtual function](#)
[virtual define](#)
[virtual expand](#)
[virtual nohide](#)
[virtual save](#)
[virtual type](#)
[virtual delete](#)
[virtual hide](#)
[virtual nolog](#)
[virtual show](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

virtual type

Creates a new enumerated type known only by the GUI, not the kernel. Converts signal values to character strings.

Syntax

```
virtual type -delete <name> | {<list_of_strings>} <name>
```

Description

The virtual type 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.

Arguments

- `-delete <name>`

(Required if not defining a type.) Deletes a previously defined virtual type.

`<name>` — The name you gave the virtual type when you originally defined it.

- `{<list_of_strings>}`

(Required if `-delete` is not used.) A list of values and their associated character strings. You can express values in decimal or based notation and can include "don't-cares" (see examples below). Supports three based notation styles: Verilog, VHDL, and C-language. Interprets values without regard to the size of the bus to be mapped. Supports bus widths up to 64 bits.

Strings that contain spaces must be enclosed in quotation marks (""). Strings that contain special characters, such as square brackets, curly braces, backslashes, and so on, must be enclosed in curly braces ({}).

See the examples below for further syntax.

- `<name>`

(Required if `-delete` is not used.) The user-defined name of the virtual type. 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 enclosed in double quotes ("") or curly braces ({}).

Examples

- Use positional notation to associate 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, and so on.

```
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
```

Related Topics

[virtual function](#)

[Virtual Objects \[ModelSim User's Manual\]](#)

vlib

Creates a design library.

Syntax

vlib -help

```
vlib [-short | -dos | -long | -unix] [-dirpath <pathname>] [-format { 1 | 3 | 4 }] [-type {directory | archive | flat}] [{-lock | -unlock} <design_unit>] [-locklib | -unlocklib] [-compress | -nocompress] <library_name>
```

Description

You must use vlib, rather than operating system commands, to create valid ModelSim library directories or index files. If a specified library already exists, the vlib command exits with a warning message, without touching the library.

Arguments

- **-compress | -nocompress**

(optional) Defines whether to store some compiled results in the library in a compressed form.

-compress — Compression occurs, producing smaller libraries. Compression can slow down subsequent executions of the vopt command.

-nocompress — (default) Library compression does not occur.

- **-dirpath <pathname>**

(optional) Specifies the path to a working directory, which is stored in the library in order to override the current working directory. This enables you hide the directory path information.

Caution

 Use of this argument is not recommended.

If you override the current working directory with -dirpath, the ModelSim user interface will be unable to find the source files when you select something in the design and ask to see the declaration.

- **-dos**

(optional) Specifies that library subdirectories have names that are compatible with DOS. Not recommended if you use the [vmake](#) utility.

On by default for ModelSim PE.

- **-format { 1 | 3 | 4 }**

(optional) Prepares a library for conversion to compatibility with a previous release, by altering the _info file.

- 1 — Converts a library to be compatible with the 6.2 series and earlier.
- 3 — Converts a library to be compatible with the 6.3 series and newer.
- 4 — Converts a library to be compatible with the 10.2 series and newer.

The usage flow is:

```
\\"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
```

- **-long**
(optional) Interchangeable with the **-unix** argument.
- **{-lock | -unlock} <design_unit>**
(optional) Locks an existing design unit so it cannot be recompiled or refreshed. The **-unlock** switch reverses this action. These switches do no affect file permissions.
- **-locklib | -unlocklib**
(optional) Locks a complete library so that compilation cannot target the library and the library cannot be refreshed. The **-unlocklib** switch reverses this action. These switches do not affect file permissions.
- **-short**
(optional) Interchangeable with the **-dos** argument.
- **-type {directory | archive | flat}**
(optional) Specifies the type of library to create.
 - directory — directory-based, legacy library. Use this option when working in a flow requiring the **vmake** command.
 - archive — archive library (replaces **vlib -archive** option).
 - flat — (default) condensed library without design unit directories.
- **-unix**
(optional) Specifies to allow subdirectories in a library to have long file names that are NOT compatible with DOS.
- **<library_name>**
(required) Specifies the pathname of the library to create.

Examples

- Create the design library, `design`. You can define a logical name for the library with the `vmap` command, or by adding a line to the library section of the `modelsim.ini` file in the same directory.

vlib design

- Creates the design library, `uut`, and specifies that any design units compiled into the library are created as archives.

vlib -type archive uut

vlog

Compiles Verilog source code and SystemVerilog extensions into either a specified working library or to the default work library. Accepts compressed SystemVerilog source files (those compressed with zlib).

Syntax

```
vlog [options] <filename> [<filename> ...]  
[options]:  
[-93]  
[-addpragmaprefix <prefix>]  
[-compat] [-compile_uselibs[=<directory_name>]] [-convertallparams] [-cuname  
<package_name>] [-cuautoname=[file | du]]  
[-define <macro_name>[=<macro_text>]] [+define+<macro_name>[=<macro_text>]] [-  
deglitchalways | -nodeglitchalways] [+delay_mode_distributed] [+delay_mode_path]  
[+delay_mode_unit] [+delay_mode_zero] [-dirpath <pathname>] [-dpiforceheader] [-  
dpifileheader <filename>]  
[-E <filename>] [-Edebug <filename>] [-enumfirstinit] [-Epretty <filename>] [-error  
<msg_number>[,<msg_number>,...]]  
[(-F | -file | -f) <filename>] [-force_refresh <design_unit>]  
[-gen_xml <design_unit> <filename>]  
[-hazards] [-help [all | category | <option> | <command-line> | <category_list>]]  
[-ignorepragmaprefix <prefix>] [+inmdir+<directory>] [-incr | -noincr] [-isymfile]  
[+iterevaluation]  
[+libcell | +nolibcell] [+libext+<suffix>] [-libmap <pathname>] [-libverbose=libmap] [-  
libmap_verbose] [+librescan] [-line <number>]  
[-lint=[default | full]] [-logfile <filename> | -l <filename>] [-lrmclassinit]  
[+maxdelays] [+mindelays] [-mixedansiports] [-mixedsvvh [b | s | v]] [-mfcu[=macro] | -sfcu] [-  
modelsimini <path/modelsim.ini>] [-msglimit {error | warning | all[,-<msgNumber>,...]} |  
none[,+<msgNumber>,...]|<msgNumber>,[msgNumber,...]] [-msglimitcount  
<limit_value> -msglimit [all,|none,] [-|+]<msgNumber>[,-|+]<msgNumber>...]]  
[-nocreatelib] [-nodbgsym]  
[-noForceUnsignedToVhdlInteger] [-nologo] [-nooverrideundef] [+nopathpulse]  
[+nospecify] [-note <msg_number>[,<msg_number>,...]] [+notimingchecks] [-  
novtblfixup] [+nowarn<CODE>] [-nowarn <category_number>]  
[-optionset <optionset_name>] [-outf <filename>] [-override_precision] [-  
override_timestep[=][ ]<time_unit> / <time_precision>] [-O0]  
[+pathpulse] [-pedanticerrors] [-permissive] [-permit_defunct_sv] [-  
printfilenames[=<filename>]]
```

```
[ -quiet ]  
[ -R [<simargs>] ] [ -refresh ]  
[ -s ] [ -sfcu ] [ -skipprotected ] [ -skipprotectedmodule ] [ -skipsynthoffregion ] [ -smartdbgsym ] [ -source ] [ -stats [=+ | -]<feature>,[,+ | -]<mode>] ] [ -suppress { <msgNumber> | <msgGroup> } ,,[<msg_number> | <msgGroup> ,...]] [ -sv ] [ -svext=[+|-]<extension>[,+[+|-]<extension>,...] ] [ -svfilesuffix=<extension>[,<extension>...]] <filename> ]  
[ -svinputport=net | var | compat | relaxed ] [ -svpkgcasesens ] [ -sv05compat ] [ -sv09compat ]  
[ -sv12compat ] [ -sv17 compat ]  
[ -timescale[=][ ]<time_units>/<time_precision> ] [ +typdelays ]  
[ -u ]  
[ -v <library_file> ] [ -version ] [ -vlog01compat ] [ -vlog95compat ] [ -vmake ]  
[ -warning <msg_number>[,<msg_number>,...]] [ -warning error ] [ -warnrbw ] [ -work <library_name> ] [ -writetoplevels <fileName> ]  
[ -y <library_directory> ]
```

Description

You can invoke the vlog command from ModelSim or from the operating system command prompt. You can also invoke it during simulation.

Compiled libraries are major-version dependent. When moving between major versions, you must use the vlog -refresh argument to refresh compiled libraries. This is not true for minor versions (letter releases).

All arguments to the vlog command are case sensitive: -WORK and -work are not equivalent.

SystemVerilog requires that the vlog command default to treating each Verilog design file listed on the command line as a separate compilation unit. To treat multiple files listed in a single command line as a single compilation unit, use either the vlog -mfcu argument or the MultiFileCompilationUnit *modelsim.ini* file variable. Refer to [MultiFileCompilationUnit](#) in the User's Manual.

Arguments

- -93
 - (optional) Specifies that the VHDL interface to Verilog modules use VHDL 1076-1993 extended identifiers to preserve the case of Verilog identifiers that contain uppercase letters.
- -addpragmaprefix <prefix>
 - (optional) Enables recognition of pragmas with a user-specified prefix. If you do not specify this argument, pragmas are treated as comments.

All regular synthesis pragmas are honored.

<prefix> — Specifies a user-defined string. The default is no string, indicated by quotation marks (" ").

You can also use the AddPragmaPrefix variable in the vlog section of the *modelsim.ini* file to set the prefix. Refer to [AddPragmaPrefix](#) in the User's Manual.

- -compat

(optional) Disables optimizations that result in different event ordering than Verilog-XL.

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 enables you to ignore them. This option does not account for all event order discrepancies, and using it may degrade performance. Refer to “[Event Ordering in Verilog Designs](#)” in the User's Manual for additional information.

- -compile_uselibs[=<directory_name>]

(optional) Locates source files specified in a `uselib directive (Refer to “[Verilog-XL uselib Compiler Directive](#)” in the User's Manual), compiles those files into automatically created libraries, and updates the *modelsim.ini* file with the logical mappings to the new libraries. If you do not specify a *directory_name*, 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.

- -convertallparams

(optional) Enables converting parameters that are not defined in ANSI style to VHDL generics of type std_logic_vector, bit_vector, std_logic, vl_logic, vl_logic_vector, and bit.

- -cuname <package_name>

(optional) Used only in conjunction with -mfcu. The -cuname argument names the compilation unit (*package_name*) being created by vlog. You can then specify the named compilation unit on the vsim command line, along with the <top> design unit. The purpose of doing so is to force elaboration of the 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
```

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. If a module that depends on that compilation unit exists, you do not need to force the elaboration, for it occurs automatically. If you are using qverilog to compile and simulate the design, this binding issue is also automatically handled properly.

- -cuautoname=[file | du]

(optional) Specifies the method for naming \$unit library entries.

file — (default) Base the name on first file in on the command line.

du — Base the name on the first design unit following items found in the \$unit scope.
Use this option when you have multiple vlog command lines that specify the same file as the first entry.

- -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>
```

This argument differs from +define, in that it allows you to use the plus (+) symbol within the macro.

- +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>
```

Optionally, 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. It also overrides all `undef directives in the RTL code — that is, any `undef for that macro is ignored. Use the **-nooverrideundef** option for backward compatibility with previous operation. If a macro is defined with the +define command line option and you use the -nooverrideundef option, the `undef is honored for that macro.

- -deglitchalways | -nodeglitchalways

Controls the behavior related to zero-delay oscillations among always_comb and always @* combinatorial logic blocks, as well as regular always blocks, that produce glitches on the variables they write.

-deglitchalways — (default) Reduces the incidents of zero delay oscillations among the affected blocks.

-nodeglitchalways — Disables the functionality. A side effect of this behavior is that time zero races involving the glitch-producing always blocks may resolve in a different order.

- +delay_mode_distributed

(optional) Disables path delays in favor of distributed delays. Refer to “[Delay Modes](#)” in the User’s Manual for details.

- +delay_mode_path

(optional) Sets distributed delays to zero in favor of using path delays.

- +delay_mode_unit

(optional) Sets path delays to zero and non-zero distributed delays to one time unit.

- **+delay_mode_zero**
(optional) Sets path delays and distributed delays to zero.
- **-dirpath <pathname>**
(optional) Specifies the path to a working directory, which is stored in the library in order to override the current working directory. This allows you hide the directory path information.

Caution



Use of this argument is not recommended.

If you override the current working directory with -dirpath, the ModelSim user interface will be unable to find the source files when you select something in the design and ask to see the declaration.

- **-dpiforceheader**
(optional) Forces the generation of a DPI header file, even if it will be empty of function prototypes.
- **-dpiheader <filename>**
(optional) Generates a header file that you can then include in C source code for DPI import functions. Refer to “[DPI Use Flow](#)” in the User’s Manual for additional information.
- **-E <filename>**
(optional) Captures text processed by the Verilog parser after preprocessing has occurred, and copies that text to an output file. This includes text read from source files specified with the -v or -y argument.

<filename> — Specifies a name for the debugging output file. You cannot use wildcards.

Generally, preprocessing consists of the following compiler directives: `ifdef, `else, `elsif, `endif, `ifndef, `define, `undef, `include.

The `line directive attempts to preserve line numbers, file names, and level in the output file (per the 1800-2009 LRM). White space is usually preserved, but is sometimes deleted or added to the output file.

- **-Edebug <filename>**
(optional) Captures text processed by the Verilog parser after preprocessing has occurred, and copies that text to a debugging output file.

<filename> — Specifies a name for the debugging output file. You cannot use wildcards.

Generally, preprocessing consists of the following compiler directives: `ifdef, `else, `elsif, `endif, `ifndef, `define, `undef, `include. The file is a concatenation of source files with `include expanded. The file can be compiled and then used to find errors in the original source files. The `line directive attempts to preserve line numbers and file names in the

output file. White space is usually preserved, but is sometimes deleted or added to the output file.

- **-enumfirstinit**

(optional) Initializes enum variables in SystemVerilog using the leftmost value as the default. You must also use the argument with the vsim command in order to implement this initialization behavior. Specify the EnumBaseInit variable as 0 in the *modelsim.ini* file to set this as a permanent default.

- **-Epretty <filename>**

(optional) 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>.

- **-error <msg_number>[,<msg_number>,...]**

(optional) Changes the severity level of the specified message(s) to "error." Edit the error variable in the *modelsim.ini* file to set a permanent default. Refer to [error](#) and "[Message Severity Level](#)" in the User's Manual for more information.

- **(-F | -file | -f) <filename>**

(optional) -f, -file and -F: each specifies an argument file with more command-line arguments, allowing you to reuse complex argument strings without retyping. Allows nesting of -F, -f and -file commands. Allows gzipped input files.

With -F only: relative file names and paths within the arguments file <filename> are prefixed with the path of the arguments file when lookup with relative path fails. Refer to "[Argument Files](#)" on page 28 for more information.

- **-force_refresh <design_unit>**

(optional) Forces the refresh of all specified design units. Default is to update the work library; 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/...
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>`

(optional) Produces an XML-tagged file containing the interface definition of the specified module. This option requires a two-step process:

- a. Compile `<filename>` into a library with vlog (without `-gen_xml`).
- b. Execute vlog with the `-gen_xml` switch.

For example:

```
vlib work
vlog counter.v
vlog -gen_xml counter counter.xml
```

- `-hazards`

(optional) Detects event order hazards involving simultaneous reading and writing of the same register in concurrently executing processes. You must also specify this argument when you simulate the design with [vsim](#). Refer to “[Hazard Detection](#)” in the User’s Manual for more details.

Note

 Enabling `-hazards` implicitly enables the `-compat` argument. As a result, using this argument can affect your simulation results.

- `-help [all | category | <option> | <command-line> | <category_list>]`

(optional) Provides online command help.

`all` — List all categories and options.

`category` — List all command categories.

`<option>` — Show help for the listed command option.

`<command-line>` — List all options for a particular category. vlog categories are:

- General
- Analog
- Coverage
- Cpp
- Debug
- GLS
- Language
- Library
- Messages
- Optimize

- **-ignorepragmaprefix <prefix>**
(optional) Directs vlog to ignore pragmas with the specified prefixname. All affected pragmas are treated as regular comments. Edit the `IgnorePragmaPrefix` *modelsim.ini* variable to set a permanent default. Refer to [IgnorePragmaPrefix](#) in the User's Manual.
 <prefix> — Specifies a user defined string.
- **-ignoresvkeywords= <keyword>[,<keyword>]...**
Instructs the complier to ignore the specified SystemVerilog keywords when encountered in a file with a .v extension, and return them as an identifier.
 <keyword>[,<keyword>]... — A comma-separated list of SystemVerilog keywords.
- **+incdir+<directory>**
(optional) Specifies directories to search for files included with `include compiler directives. By default, searches the current directory first and then the directories specified by the +incdir options, in the order they appear on the command line. You can specify multiple +incdir options and multiple directories, separated by "+", in a single +incdir option.
- **-incr**
(optional) Performs an incremental compilation. Compiles only code that has changed. For example, if you change only one module in a file containing several modules, only the changed module is recompiled. However, if you change the compile options, all modules are recompiled, regardless of whether you use vlog -incr or not.
- **-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.
- **+iterevaluation**
(default) Enable an iterative evaluation mechanism on optimized gate-level cells with feedback loops.
- **-logfile <filename> | -l <filename>**
(optional) Generates a log file of the compile.
 -logfile <filename> — Saves transcript data to <filename>. Can be abbreviated to -l <filename>. Overrides the default transcript file creation set with the TranscriptFile or BatchTranscript File *modelsim.ini* variables (refer to [TranscriptFile](#) or [BatchTranscriptFile](#) in the User's Manual.) You can also specify "stdout" or "stderr" for <filename>.
- **+libcell | +nolibcell**
 +libcell — (optional) Treats all modules found and compiled by a source library search as though they contain a 'celldesignate' compiler directive, and marks 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.

+nolibcell — (default) Disables treating all modules found and compiled by source library search as though they contained a ‘celldefine compiler directive. This argument restores the default library search behavior after you have used the [+libcell](#) | [+nolibcell](#) argument to change it.

- **+libext+<suffix>**
(optional) Works in conjunction with the **-y** option. Specifies file extensions for the files in a source library directory. By default, the compiler searches for files without extensions. If you specify the **+libext** argument, the compiler searches for a file with the suffix appended to an unresolved name. You can specify only one **+libext** option, but it can contain multiple suffixes separated by the plus character (+). The extensions are tried in the order you specify them with the **+libext** argument.
- **-libmap <pathname>**
(optional) Specifies a Verilog 2001 library map file. You can omit this argument by placing the library map file as the first option in the **vlog** invocation (for example, *vlog top.map top.v top_cfg.v*). You can use the **vlog -mfcu** argument to compile macros for all files in a given testbench. Any macros already defined before the **-libmap** argument appears are still defined for use by the **-libmap** files.
- **-libverbose=libmap**
(optional)
Displays library map pattern matching information during compilation. Use this argument to troubleshoot problems with matching filename patterns in a library map file. For example, when a resolved module has a choice between two libraries, this argument tells you which one was selected, confirming that your config file worked.
- **-libmap_verbose**
(optional) Displays library map pattern matching information during compilation.

Note

 This argument is being deprecated — the recommended method is to use the **-libverbose=libmap** argument instead. However, for compatibility reasons, **-libmap_verbose** will continue to be supported indefinitely, so it is safe to use until further notice.

- **+librescan**
(optional) Scans libraries in command-line order for all unresolved modules.
- **-line <number>**
(optional) Starts the compiler on the specified line in the Verilog source file. By default, the compiler starts at the beginning of the file.
- **-lint=[default | full]**
(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.

default — Performs static checks and generates code for vsim, but has no impact on vsim performance as checks are not deferred to runtime.

full — Performs static checks in vlog and vopt and defers checks to vsim if required. If checks are deferred to vsim, simulation run time will be impacted.

The “full” option 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.

You can also enable this option with the Show_Lint variable in the *modelsim.ini* file. Refer to [Show_Lint](#) in the User’s Manual.

- **-lrmclassinit**

Changes initialization behavior to match the SystemVerilog specification (per IEEE Std 1800-2007), where all superclass properties are initialized before any subclass properties.

- **+maxdelays**

(optional) Selects maximum delays from the "min:typ:max" expressions. You can defer delay selection until simulation time by specifying the same option to the simulator.

- **+mindelays**

(optional) Selects minimum delays from the "min:typ:max" expressions. You can defer delay selection until simulation time by specifying the same option to the simulator.

- **-mfcu[=macro]**

(optional) Instructs the compiler to treat all files within a compilation command line as a single compilation unit. The default behavior is to treat each file listed in a command as a separate compilation unit, as is the SystemVerilog standard. Prior versions concatenated the contents of the multiple files into a single compilation unit by default. When specified, the =macro modifier enables the visibility of macro definitions across different files.

All global declarations present in both compile file and library files specified with the -v argument are lumped together in a single \$unit scope.

You can use -mfcu to compile macros for all files in a given testbench. Any macros already defined before the -libmap argument appears are still defined for use by the -libmap files.

You can also use the MultiCompilationUnit variable in the *modelsim.ini* file to enable this option (without the =macro functionality). Refer to [MultiFileCompilationUnit](#) in the User’s Manual.

- **-mixedansiports**

Use this switch only when your design files contain a combination of ANSI and non-ANSI port declarations and task/function declarations. For example:

```
module top (input reg [7:0] a,
            output b);
    reg [7:0] b;
endmodule
```

- **-mixedsvvh [b | s | v]**

(optional) Facilitates using SystemVerilog packages at the SystemVerilog-VHDL boundary of a mixed-language design. When you compile a SystemVerilog package with -mixedsvvh, you can include the package in a VHDL design as if it were defined in VHDL itself.

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

The simulator converts any case sensitivity in the SystemVerilog package to extended VHDL identifiers for local params. For example, *Variable1* in a system Verilog package is converted to *Variable1* in the equivalent VHDL package.

- **-modelsimini <path/modelsim.ini>**

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, including the file name. On Windows systems the path separator is a forward slash (/).

- **-msglimit {error | warning | all[,-<msgNumber>,...] | none[,+<msgNumber>,...] | <msgNumber>,[msgNumber,...]}**

(optional) Limits messages to the default message limit count of five. Use the -msglimitcount argument to change the default message limit count. Specify multiple messages as a comma-separated list.

error — Stops the run when the total number of errors reaches the default count.
warning — Stops the run when the total number of warnings reaches the default count.
all — Limits all messages to the default count except those you specifically exclude. To exclude messages from the limit, supply a comma-separated list of message numbers, each preceded by a minus sign (-), for example “all,-<msgNumber1>,<msgNumber2>,...”.

none — Excludes all messages from the default count except those you specifically include. To include messages to limit to the default count, supply a comma-separated list of message numbers, each preceded by a plus sign (+), for example “none,+<msgNumber1>,+<msgNumber2>,...”.

<msgNumber>[,<msgNumber>,...] — Specifies messages to limit to the default count.

Examples:

- Stop the run when the total number of errors reaches the default count:

```
-msglimit error
```

- Stop the run when the total number of warnings reaches the default count:

```
-msglimit warning
```

- Specifically limit messages 2374 and 2385 to the default count:

```
-msglimit 2374,2385
```

- Limit all messages to the default count, except messages 2374 and 2385:

```
-msglimit all,-2374,-2385
```

- Limit only messages 2374 and 2385 to the default count:

```
-msglimit none,+2374,+2385
```

- **-msglimitcount <limit_value>** -msglimit [all,none,] [-+]<msgNumber>[,-+]<msgNumber>...]

(optional) Limits the reporting of listed messages to user-defined limit_value. Overrides the MsgLimitCount variable in the modelsim.ini file. Refer to [MsgLimitCount](#) in the User's Manual.

- **-nocreatelib**

(optional) Stops automatic creation of missing work libraries. Overrides the CreateLib modelsim.ini variable. Refer to [CreateLib](#) in the User's Manual.

- **-nodbgsym**

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 enabling you to view detailed information about design objects at the source level. Two major GUI features that use this database are source window annotation and textual dataflow.

You should specify this switch only if you know that no one else using the library will require this information for design analysis purposes.

- **-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*3 -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).

- **-noincr**
(optional) Disables incremental compilation previously turned on with -incr argument.
Default.
- **-nologo**
(optional) Disables the startup banner.
- **-nooverrideundef**
(optional) Prevents `undefs from being overridden by macros defined with the +define command line option. If a macro is defined with +define command line option, and -nooverrideundef is also passed as a compile option, the `undef is honored for that macro.
- **+nopathpulse**
(optional) Causes PATHPULSE\$ specparam(s) to be ignored in a module's specify block. The default is to ignore PATHPULSE\$ specparam(s).
- **+nospecify**
(optional) Removes specify blocks from compiled Verilog cells and modules. Causes \$sdf_annotation() to be ignored.

You should generally pass this argument to vopt in the three step flow, or vsim in the two-step flow instead of to vlog. This is because passing +nospecify to vlog removes specify blocks from the compiled Verilog cells and modules, meaning you must recompile the libraries to run a timing simulation at a later point.
- **-note <msg_number>[,<msg_number>,...]**
(optional) Changes the severity level of the specified message(s) to "note." Edit the note variable in the *modelsim.ini* file to set a permanent default. Refer to [note](#) and "[Message Severity Level](#)" in the User's Manual for more information.
- **+notimingchecks**
(optional) Removes all timing check entries from the design as it is parsed. You should generally pass this argument to vopt in the three step flow, or vsim in the two-step flow instead of to vlog. This is because passing +notimingchecks to vlog removes timing checks from the compiled Verilog cells and modules, meaning you must recompile the libraries to run a timing simulation at a later point.
- **-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 the default behavior of treating the type of a this reference 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 will not execute methods of an uninitialized class type).

- **+nowarn<CODE>**

(optional) Disables warning messages in the category specified by <CODE>; those warnings that include the <CODE> name in square brackets in the warning message. For example, +nowarnRDGN disables the following warning message:

```
** Warning: test.v(15): [RDGN] - Redundant digits in numeric literal.
```

- **-nowarn <category_number>**

(optional) Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message to suppress. You can include multiple -nowarn switches. Warnings can be disabled for all compiles via the Main window **Compile > Compile Options** menu command, or the *modelsim.ini* file (refer to [modelsim.ini Variables](#) in the User's Manual).

The following table describes the warning message categories:

Table 2-8. Warning Message Categories for vlog -nowarn

Category number	Description
12	non-LRM compliance in order to match Cadence behavior

- **-optionset <optionset_name>**

(optional) Calls an optionset as defined in the *modelsim.ini* file. Refer to “[Optionsets](#)” for more information.

- **-outf <filename>**

(optional) Specifies a file to save the final list of options to, after recursively expanding all -f, -file and -F files.

- **-override_precision**

(optional) Used with the -timescale argument, this argument overrides the precision of `timescale specified in the source code.

- **-override_timescale[=][]<time_unit> / <time_precision>**

(optional) Specifies a timescale for all compiled design units. This timescale overrides all ‘timescale directives and all declarations of timeunit and timeprecision. You can use an equal sign (=) or a space between option and arguments.

time_unit — Unit of measurement for times and delays. This specification consists of one of three integers (1, 10, or 100) representing order of magnitude, and one of six character strings representing units of measurement: {1 | 10 | 100} {s | ms | us | ns | ps | fs}. For example, 10 ns.

time_precision — Unit of measurement for rounding delay values before using them in simulation. Allowable values are the same as for time_unit.

- **-O0**
(optional) Lower the optimization to a minimum with -O0 (capital oh zero). 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.
- **+pathpulse**
(optional) Enables usage of the PATHPULSE\$ specparam in a module's specify block. Has no effect on modules without PATHPULSE\$ specparam(s). The default is to ignore PATHPULSE\$ specparam(s).
- **-pedanticerrors**
(optional) Enforces strict compliance of the IEEE Std 1800-2005. The following are some of the 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 occurs 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 occurs for literals written without the grave accent mark.
 - Inserting the grave accent mark to precede quotation marks (`") that enclose string literals is not legal—only string literals within quotation marks ("") are allowed. When you specify this argument, an error occurs for string literals using that mark.
 - Using class extern method prototypes with lifetime (automatic/static) designations produces a compliance error (instead of a warning).
 - Using “cover bool@clk” as a PSL statement is not legal.
 - Using an unsized constant in a concatenation if it is the leftmost value in the list is not legal.
 - Calling a virtual function in the constructor of the same class is not legal.
 - Using integers to define macro names is not legal.
 - Declaring *unique* constraint as *soft* is not LRM-complaint.
 - Using non-LRM compliant extensions. When you specify this argument, it disables all non-LRM compliant extensions.

This argument also produces a report of mismatched ‘else’ directives.

You can produce a complete list by executing the command:

```
verror -kind vlog -pedanticerrors
```

- **-permissive**

(optional) Allows messages in the LRM group of error messages to be downgraded to a warning. Allows the use of reserved keywords 'config' and 'instance' outside of unit and configuration scopes. Also allows named port connections on bit-select and part-select ports, though only when multiple bit-select or part-select ports of same name are not present in the port list.

You can produce a complete list by executing the command:

```
verror -kind vlog -permissive
```

- **-permit_defunct_sv**

(optional) 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.

- **-printinfilenames[=<filename>]**

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. To write these path names to a text file in the current directory, add =<filename> to this argument. If you use this argument again with the same filename, you overwrite the contents of the previous version of the file.

- **-quiet**

(optional) Disables output of informative messages about processing the design, such as Loading, Compiling, or Optimizing, but not messages about the design itself.

- **-R [<simargs>]**

Instructs the compiler to invoke **vsim** after compiling the design. The compiler automatically determines which top-level modules are to be simulated.

When you use the -R option, you must specify the log files for vlog and vsim separately. The file you specify before -R will capture the output of the vlog compiler, and the one you specify after -R will capture the vsim output.

For example, in the following vlog command, "log1.txt" will contain the vlog output and "log2.txt" will contain the vsim output.

```
vlog -l log1.txt top.sv -R -c -do "run -all;quit" -l log2.txt
```

The -R option is not a Verilog-XL option, but ModelSim uses it to combine the compile and simulate phases together, as you may be used to doing with Verilog-XL. This option is provided to ease the transition to ModelSim. It is not recommended that you use this option regularly, as you will then incur the unnecessary overhead of compiling your design for each simulation run.

- **-refresh**
(optional) Regenerates a library image. 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 can use a specific design name with `-refresh` to regenerate a library image for that design, but you cannot use a file name.
- **-s**
(optional) Instructs the compiler not to load the standard package. Use this argument only when you are compiling the `sv_std` package.
- **-sfcu**
Instructs the compiler to treat all files in a compilation command line as a separate compilation units. This is the default behavior, and is the inverse of the behavior of `-mfcu[=macro]`.
If this file has global declarations, a local \$unit scope is created for a library file passed through with the `-v` argument.
This switch overrides the `MultiFileCompilationUnit` variable if it is set to "1" in the `modelsim.ini` file. Refer to [MultiFileCompilationUnit](#) in the User's Manual.
- **-skipprotected**
(optional) Ignores any 'protected'/'endprotected region contained in a module.
- **-skipprotectedmodule**
(optional) Prevents adding any module containing a 'protected'/'endprotected region to the library.
- **-skipsynthoffregion**
(optional) Ignore all constructs within `synthesis_off` or `translate_off` pragma regions.
- **-smartdbgsym**
(optional) Reduces the size of design libraries by minimizing the number of debugging symbol files generated at compile time.
Edit the `SmartDbgSym` variable in the `modelsim.ini` file to set a permanent default. Refer to [SmartDbgSym](#) in the User's Manual.
- **-source**
(optional) Displays the associated line of source code before each error message generated during compilation. The default is to display only the error message.
- **-stats [=+ | -]<feature>[,[+ | -]<mode>]**
(optional) Controls display of compiler statistics sent to a logfile, `stdout`, or the transcript. Specifying `-stats` without options sets the default features (`cmd`, `msg`, and `time`).

Specify multiple features and modes for each instance of -stats as a comma-separated list. You can specify -stats multiple times on the command line, but only the final instance takes effect.

[+ | -] — Controls activation of the feature or mode, where the plus character (+) enables the feature and the minus character (-) disables the feature. You can also enable a feature or mode by specifying it without the plus (+) character. Setting this switch adds or subtracts features and modes from the settings in the Stats *modelsim.ini* variable.

<feature>

all — Display all statistics features (cmd, msg, perf, time). Mutually exclusive with none option. When specified in a string with other options, all is applied first.

cmd — (default) Echo the command line.

msg — (default) Display error and warning summary at the end of command execution.

none — Disable all statistics features. Mutually exclusive with all option. When specified in a string with other options, none is applied first.

perf — Display time and memory performance statistics.

time — (default) Display Start, End, and Elapsed times.

Note

 Use -stats=perf+verbose to display information about the operating system and host machine.

<mode>

Modes can be set for a specific feature or globally for all features. To add or subtract a mode for a specific feature, specify using the plus (+) or minus (-) character with the feature, for example, vlog -stats=cmd+verbose, perf+list. To add or subtract a mode globally for all features, specify the modes in a comma-separated list, for example, vlog -stats=time,perf,list,-verbose. You cannot specify global and feature specific modes together.

kb — Print performance statistics in kilobyte units with no auto-scaling.

list — Display statistics in a Tcl list format when available.

verbose — Display verbose statistics information when available.

Note

 vlog -quiet disables all default or user-specified -stats features.

- -suppress {<msgNumber> | <msgGroup>} {[,<msg_number> | <msgGroup>],...]
(optional) Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message to suppress. You cannot suppress Fatal or Internal messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to [suppress](#) and “[Message Severity Level](#)” in the User’s Manual for more information.

<msgNumber>

A specific message number

<msgGroup>

A value identifying a pre-defined group of messages, based on area of functionality. These groups only suppress notes or warnings. The valid arguments are:

All, GroupNote, GroupWarning, GroupFLI, GroupPLI, GroupSDF, GroupVCD, GroupVital, GroupWLF, GroupTCHK, GroupPA, GroupLRM

- -sv

(optional) Enables SystemVerilog features and keywords, including functional coverage constructs. By default ModelSim follows the IEEE Std 1364-2005 and ignores SystemVerilog keywords. If a source file has a ".sv" extension, ModelSim automatically parses SystemVerilog keywords.

- -svext=[+|-]<extension>[,[+|-]<extension>]...

(optional) Enables or disables non-LRM compliant SystemVerilog language extensions with a comma-separated list of arguments.

[+ | -] — Controls activation of the *extension*. Remember that command arguments take precedence; any settings to this argument override your settings of the SV Extensions *modelsim.ini* variable. Refer to [SvExtensions](#) in the User's Manual.

+ — Activates the *extension*.

- — Deactivates the *extension*.

If you do not specify either a "+" or "-", the command assumes you are activating the specified *extension*.

<extension> —

Note



Specify multiple extensions as a comma-separated list. For example:

vlog -svext=+fec1,-uslt,pae

acum — Specifies that the get(), try_get(), peek(), and try_peek() methods on an untyped mailbox will return successfully if the argument passed is assignment-compatible with the entry in the mailbox. The LRM-compliant behavior is to return successfully only if the argument and entry are of equivalent types.

acade — Adds always_comb driven elements in the sensitivity lsit.

adsl — Adds dynamic element (class element access) to the sensitivity list of "always @*". Null class handles can lead to crash at elaboration or run time.

altsic — Allows local lookup of "this" and "super" in inline constraints.

alpo — Allows overriding local parameters using the -G argument in vopt. It cannot be used to override class parameters.

altdpiheader — Allows an alternative style function signature generated in a DPI header.

ared — Allows use of array reduction methods on multi-dimensional unpacked arrays, without the need of using a 'with' clause. A multi-dimensional unpacked array will be treated as if it had a single dimension [0:total_number_of_elements-1].

arif — Allows the use of refs in fork-join_any or fork-join_none blocks inside tasks.

aswe — Enables support for the symmetric wild equality operators `=?=` and `!?=`.

atpi — Uses type names as port identifiers. Disabled when compiling with `-pedanticerrors`.

bstr — Allow use of the string built-in method `bittosstr`.

catx — Allows an assignment of a single unsized constant in a concat to be treated as an assignment of 'default:val'.

ctlc — Casts time literals in constraints to the type: time. The LRM dictates that a time literal, such as "10ns" is to be interpreted as a "realtime" type, but this is not always adhered to, for example:

```
class Foo;
    rand time t;
    constraint c1 {
        t < 10ns; // NON-LRM compliant use of 'real' type
    }
endclass
```

daoa — Allows the passing a dynamic array as the actual argument of DPI open array output port. Without this option, a runtime error, similar to the following, is generated, which is compliant with LRM requirement.

```
# ** Fatal: (vsim-2211) A dynamic array cannot be passed as an
argument to the DPI import function 'impcall' because the formal 'o'
is an unsized output.
#   Time: 0 ns  Iteration: 0  Process: /top/#INITIAL#56 File:
dynarray.sv
# Fatal error in Module dynarray_sv_unit at dynarray.sv line 2
```

dbpp — Decrypt before preprocessing.

ddup — (Drive Default Unconnected Port) Reverts behavior to where explicit named unconnected ports are driven by the default value of the port.

defervda — Causes SystemVerilog variables that have an initializer in the declaration to trigger top-blocking always blocks at time zero.

dmsbw — Drives the MSB unconnected bits of the wider hiconn (output) or the wider loconn (input) in an otherwise collapsible port connection. With this extension, zero drives unconnected bits; otherwise, they float.

evdactor — Enables early variable declaration assignments during class construction. The default behavior is to perform all superclass initialization before initializing any fields in a subclass.

evis — Supports the expansion of environment variables in curly braces ({}) in `include string literals and in `include path names. For example, if MYPATH exists in the environment then it will be expanded in the following:

`include "\$MYPATH/inc.svh"

feci — Treats constant expressions in a foreach loop variable index as constant.

fin0 — Treats \$finish() system call as \$finish(0), which results in no diagnostic information being printed.

hiercross1 — Allow hierarchical cross feature.

hiercross2 — Allow an alternate hierarchical cross feature.

ias — Iterates on always @* evaluations until inputs settle. Typically, an always @* block is not sensitive to events generated by executing the block itself. This argument increases the sensitivity of the block, so that it will re-trigger if any input has changed since the last iteration of the always block.

idcl — Allows passing of import DPI call locations as implicit scopes.

When there is no svSetScope() call, the DPI scope will always be implicitly set to the current DPI call location and restored after the call returns. This is different than the LRM behavior, which sets the scope to the import function declaration's scope instead of the call location.

When you call svSetScope in a C function, it stays sticky until the enclosing C function returns. The calling chain initiates inside the enclosing C function, but after svSetScope(), will always see the same user scope setting. svSetScope() will not impact additional DPI calls in the same thread after the enclosing C function returns.

You can apply this argument to a subset of design hierarchies within the same simulation. Design units optimized with, and without idcl can also coexist within the same vsim session. Apply this argument with caution, and only for design hierarchies that require the non-LRM behavior.

iddp — Ignores the DPI task disable protocol check.

ifca — Iterate on feedback continuous assignment until inputs settle.

ifslvbefr — Allows a solve/before constraint within an IfElse constraint with a constant condition. This is on by default.

islm — Ignore string literals within macros.

lrtic — Use return type declared inside class for extern function, without scope resolution.

mewq — Allows macro substitution inside string literals not within a text macro.

mncim — Allows macro names to contain minus ('-') characters.

mtdl — Ignores commas and right parentheses occurring within a `` `` (back tick-double quotation mark back tick-double quotation mark) string that forms a list of macro argument actuals within another macro expansion.

mttd — Reinterprets `` `` (back tick-back tick-double quotation mark) as `` `` (back tick-double quotation mark) within macro text expansion.

ncref — Prevents a ref argument in the new operator of a covergroup to be treated as a constant, unless specified.

noexptc — Ignore DPI exports of SystemVerilog type name overloading check.

omaa — Allows shuffle ordering method on associative arrays.

pae — Automatically export all symbols imported and referenced in a package.

pae1 — Allows the export, using wildcard export only, of package symbols from a subsequent import of a package. These symbols may or may not be referenced in the exporting package.

pctap — Promote concat to an assignment pattern using heuristics, such as the presence of unsized literals.

qcat — Allows the use of an assignment pattern for concatenating onto a queue.

sas — Enables LRM-compliant @* sensitivity rules.

sccts — Processes string concatenations converting the result to string type.

sceq — Allows string comparison with SystemVerilog case equality operator (==).

sref — Allows you to specify ref arguments in covergroup sample functions without using const.

sffi — Allows the same function and formal identifier in the case of functions with a void return type.

softunique — Enables you to declare the *unique* constraint as *soft*.

spsl — (default) Searches for packages in source libraries specified with -y and +libext.

stop0 — Treats \$stop and \$stop() as \$stop(0), which results in no diagnostic information being printed.

substr1 — Allows one argument in the builtin function substr. A second argument will be treated as the end of the string.

ubdic — Allows the use of a variable in a SystemVerilog class before it is defined. For example:

```
class A;
    function func();
        int y = x;      // variable 'x' is used before it is defined
    endfunction
    int x = 1;
endclass
```

If you do not enable this extension, you will receive unresolved reference errors.

udm0 — Expands any undefined macro with the text “1'b0”.

uslt — (default) Promotes unused design units found in source library files specified with the -y option to top-level design units.

vmctor — Allows virtual method calls in class constructor. The default is to treat them as non-virtual during construction.

voidsystf — When enabled (default), specifies to evaluate any occurrences of \$void(x) within constraint contexts as (x), and issue a suppressible warning each time. Occurrences of \$void outside of constraint contexts will behave as user-defined system function calls (PLI).

When disabled, evaluates any occurrences of \$void(x) as user-defined system function calls (PLI). Because system function calls are not generally allowed within constraint contexts, the use of \$void in a constraint generates a vlog/vopt error.

Specifying -pedanticerrors disables the voidsystf extension, even if you enable it with -svext=voidsystf.

- **-svfilesuffix=<extension>[,<extension>...]** <filename>

Allows specification of filename extensions for SystemVerilog files. Overrides the SVFileSuffixes variable in the *modelsim.ini* file for specified <filename>. Refer to **SVFileSuffixes** in the User's Manual.

- **-svinputport=net | var | compat | relaxed**

(optional) Used in conjunction with -sv (or files with an .sv extension) to determine the default kind for an input port that is declared with a type, but with out the var keyword.

net — declares the port to be a net. This value enforces strict compliance to the Verilog LRM (IEEE Std 1364-2005), where the port declaration defaults to the default net type as specified by the 'default_nettype directive.

var — declares the port to be a variable. This value enforces behavior from previous releases, where the port declaration defaults to variable.

compat — declares that only types compatible with net declarations default to wire.

relaxed — (default) declares the port to be a net only if the type is a 4-state scalar or 4-state single dimensional vector. Otherwise, the port is declared a variable.

- **-svpkgcasesens**

(optional) Requires case-sensitive matching between SystemVerilog package import statements and package names.

- **-sv05compat**

Used in conjunction with the -sv switch to ensure compatibility with the reserved keyword set of IEEE Std 1800-2005.

- **-sv09compat**

Used in conjunction with the -sv switch to ensure compatibility with the reserved keyword set of IEEE Std 1800-2009.

- **-sv12compat**

Used in conjunction with the -sv switch to ensure compatibility with the reserved keyword set of IEEE Std 1800-2012.

- **-sv17 compat**
Used in conjunction with the -sv switch to ensure compatibility with the reserved keyword set of IEEE Std1800-2017
- **-timescale[=][]<time_units>/<time_precision>**
(optional) Specifies the default timescale for all design unit types (modules, interfaces, programs, packages, checkers, and so forth) not having an explicit timescale directive in effect during compilation.
The format of the -timescale argument is the same as that of the `timescale directive. An equal sign (=) or whitespace is accepted between option and arguments. If you use a space between arguments, you must enclose <time_units> / <time_precision> in quotation marks ("). 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_precision> must be smaller than or equal to the <time_units>. Refer to “[Simulator Resolution Limit \(Verilog\)](#)” in the User’s Manual for more information.
- **+typdelays**
Selects typical delays from the "min:typ:max" expressions. Default. You can specify the same option to the simulator to defer delay selection until simulation time.
- **-u**
(optional) Converts regular Verilog identifiers to uppercase. Allows case insensitivity for module names.
- **-v <library_file>**
(optional) Specifies a source library file containing module and UDP definitions. Refer to “[Verilog-XL Compatible Compiler Arguments](#)” in the User’s Manual 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. You can have multiple -v options. See additional discussion in the examples.
- **-version**
(optional) Returns the version of the compiler as used by the licensing tools.
- **-vlog01compat**
(optional) Ensures compatibility with rules of IEEE Std 1364-2001.
- **-vlog95compat**
(optional) 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. Edit the vlog95compat variable in the *modelsim.ini* file to set a permanent default. Refer to [vlog95compat](#) in the User’s Manual.

- **-vmake**

Generates a complete record of all command line data and files accessed during the compile of a design. This data is 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>,...]**

(optional) Changes the severity level of the specified message(s) to "warning." Edit the warning variable in the *modelsim.ini* file to set a permanent default. Refer to [warning](#) and "[Message Severity Level](#)" in the User's Manual for more information.

- **-warning error**

(optional) Reports all warnings as errors.

- **-warnrbw**

(optional) Displays a warning when a variable is read before written in an always @* block.

- **-work <library_name>**

(optional) Specifies a logical name or pathname of a library that is to be mapped to the logical library work. 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. Can be specified only when compiling the top level modules.

<fileName> — Required. Specifies the name of the file in which to record module names.

- **-y <library_directory>**

(optional) Specifies a source library directory containing definitions for modules, packages, interfaces, and user-defined primitives (UDPs). Typically, this is a directory of source files to scan if the compiled versions do not already exist in a library. Refer to "[Verilog-XL Compatible Compiler Arguments](#)" in the User's Manual for more information.

After processing all explicit filenames on the vlog command line, the compiler uses the -y option to find and compile any modules that were referenced but not yet defined. Files in this directory are compiled only if the file names match the names of previously unresolved references. You can use multiple -y options. You must specify a file suffix by using -y in conjunction with the +libext+<suffix> option. 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. A filename is required. You can enter multiple filenames, separated by spaces. You can use wildcards.

Examples

- Compile the Verilog source code contained in the file *example.vlg*.

vlog example.vlg

- 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](#)” in the User’s Manual 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"

-
- Enable the display of Start, End, and Elapsed time as well as a message count summary. Echoing of the command line is disabled

vlog -stats=time,-cmd,msg

- The first -stats option is ignored. The none option disables all default settings and then enables the perf option.

vlog -stats=time,cmd,msg -stats=none,perf

vmake

Enables you to use a MAKE program to maintain individual libraries.

Note

 You must invoke this command from the system prompt. If a design is spread across multiple libraries, then each library must have its own makefile and you must build each one separately.

Syntax

```
vmake [-du <design_unit_name> ...] [-f <filename>] [-fullsrcpath] [-ignore] [<library_name>]  
[-modelsimini <path/modelsim.ini>]
```

Description

You run vmake on a compiled design library. It operates on multiple source files in the design unit, supporting Verilog include files, as well as Verilog and VHDL PSL vunit files.

By default, the output of vmake is sent to stdout. 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.

A MAKE program is included with Microsoft Visual C/C++, as well as many other program development environments.

Note

 Makefile support for flat libraries is limited because of the lack of per-target file objects. However, the resulting makefile does trigger a build of all design units if any source file for any design unit is newer than the library. Optimized design units in flat libraries are also supported, with more precise dependency tracking.

You run vmake once; then run MAKE to rebuild your design. MAKE recompiles only the design units (and their dependencies) that have changed. 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.

Arguments

- **-du <design_unit_name>**
(optional) Specifies to generate a vmake file for only the specified design unit. You can specify this argument any number of times for a single vmake command.
- **-f <filename>**
(optional) Specifies a file to read command line arguments from.
Refer to the section "[Argument Files](#)" on page 28 for more information

- **-fullsrcpath**
(optional) 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 in directories that are different than the directory where compilations originally occurred.
- **-ignore**
(optional) Omits a make rule for the named primary design unit and its secondary design units.
- **<library_name>**
(optional) Specifies the library name; if no name is specified, then work is assumed.
- **-modelsimini <path/modelsim.ini>**
(optional) Loads an alternate initialization file to replace 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, including the file name. On Windows systems, the path separator is a forward slash (/).

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
- To use vmake and MAKE on your work library:
C:\MIXEDHDL> vmake >makefile
- To edit an HDL source file within the work library:
C:\MIXEDHDL> make
Your design gets recompiled for you. You can change the design again and re-run MAKE to recompile additional changes.
- To run vmake on libraries other than work:
C:\MIXEDHDL> vmake mylib >mylib.mak
- To rebuild mylib, specify its makefile when you run MAKE:
C:\MIXEDHDL> make -f mylib.mak

vmap

Defines a mapping between a logical library name and a directory by modifying the *modelsim.ini* file.

Syntax

```
vmap [-c | -del <logical_name> ... | <logical_name> [<path>] ]  
      [-modelsimini <path/modelsim.ini>]
```

Description

With no arguments, vmap reads the appropriate *modelsim.ini* file(s) and prints to the transcript the current logical library to physical directory mappings.

Arguments

- **-c**
(optional) Copies the default *modelsim.ini* file from the ModelSim installation directory to the current directory.
Use this argument only to copy 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>.
- **-modelsimini <path/modelsim.ini>**
(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, including the file name. On Windows systems the path separator is a forward slash (/).

Examples

- Map two logical libraries to the physical library “work”:

```
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

vsim

Invokes the VSIM simulator. Invoking this command with the -view argument enables you to view the results of a previous simulation run.

Syntax

This section lists all arguments of the vsim command in alphabetical order.

The Arguments section groups the argument descriptions into the following categories:

Note

Argument Groups

- [All languages](#)
 - [VHDL Arguments](#)
 - [Verilog Arguments](#)
 - [Object Arguments](#)
-

vsim [arguments]

[arguments]:

```
[-allowcheckpointcpp 1|0] [-appendlog] [-accessobjdebug | -noaccessobjdebug]
 [+alt_path_delays] [-assertfile <filename>] [-autofindloop]
 [-batch] [+bitblast[=iopath | tcheck]]
 [-c] [-capacity[=line]] [-capstats[=<args>[,]]] [-checkvifacedrivers] [-classdebug | -
 noclassdebug] [-collapseall] [-colormap new]

 [-default_radix <radix>] [-defaultstdlogicinit0z] [+delayed_timing_checks] [-display
 <display_spec>] [-displaymsgmode both | tran | wlf] [-do "<command_string>" |
 <do_file_name>] [-donotcollapsepartiallydriven] [-dpiforceheader] [-dpiheader] [-dpilib
 <libname>] [-dpioutoftheblue 0 | 1 | 2] [+dumports+collapse | +dumports+nocollapse]
 [+dumports+direction] [+dumports+no_strength_range] [+dumports+unique]

 [-error <msg_number>[,<msg_number>,...]][-enumfirstinit]
 [-f <filename>] [-fatal <msg_number>[,<msg_number>,...]] [-fsmdebug]
 [-g <Name>=<Value> ...] [-G<Name>=<Value> ...] [-gblso <shared_obj>[,<shared_obj>]] [-
 gblsoauto] [-gconrun | -nogconrun] [-gconstep | -nogconstep] [-gcthreshold <n>] [-geometry
 <geometry_spec>] [-glsnegtchk [level0 | level1 | level2 | level3]] [-glspath simuconddc] [-
 gui]

 [-hazards]-help [all | category | <option> | <command-line> | <category_list>]
 [-i] [-ignoreinilibs] [+initregNBA | +noinitregNBA] [-installcolormap] [-
 iterationlimit[=<limit>[k][m][g] | [i[ncrease]]]]
```

```
[-keeploaded] [-keeploadedrestart] [-keepstdout]
[-logfile <filename> | -l <filename> | -nolog] [-lib <libname>] [-libverbose[=plib]]
[<library_name>.<design_unit>] [-L <library_name> ...][ -Ldir <pathname> [<pathname>
...]] [-Lf <library_name> ...]

[-modelsimini <path/modelsim.ini>] [-msgfile <filename>] [-msglimit {error | warning | all[,-
<msgNumber>,...] | none[,+<msgNumber>,...] | <msgNumber>,[msgNumber,...]}] [-
msglimitcount <limit_value> -msglimit [all, | none,] <msgNumber>[, <msgNumber>...]] [-
msglimitcount <limit_value> -msglimit [error|warning]] [-msgmode both | tran | wlf][-
msgsingleline] [-multisource_delay min | max | latest] [+multisource_int_delays]

[-name <name>] [-noappendclose] [+no_autodtc] [-noautoldlibpath] [-nodpiexports] [-
noautofindloop] [+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] [-nofileshare] [-noimmedca] [-
noglitch][+nosdferror] [+nosdfwarn] [+nospecify] [-nostdout] [-note
<msg_number>[,<msg_number>,...]] [+notifier_ondetect] [+notimingchecks |
+ntcnotchks] [-noverboseprofile] [-novhdlvariablelogging] [+nowarnBSOB]
[+nowarn<CODE | number>] [-nowiremodelforce] [+ntc_warn] [+ntcnotchks]

[-oldvhdlforgennames] [-onfinish ask | stop | exit | final] [-optionset <optionset_name>]

[-pduignore[=<instpath>]] [-pdupath[=<lib_path>]] [-pedanticerrors] [-permissive] [-
permit_unmatched_virtual_intf] [-pli "<object list>"] [+<plusarg>] [-postsimdataflow] [-
printsimstats[=<val>][v]] [+pulse_e/<percent>] [+pulse_e_style_ondetect]
[+pulse_e_style_onevent] [+pulse_r/<percent>] [+pulse_int_e/<percent>] [+pulse_int_r/
<percent>]

[-quiet] [+questa_mvc_core+wlf_disable <filename>]
-rngtrace[=<filename>] [-rngtraceopt=[+|-]<option_string>[,[+|-]<option_string>]] [-
rngtraceclassfilter=<class_filter_string>[,<class_filter_string>]] [-
rngtraceprocessfilter=<scope_filter_string>[,<scope_filter_string>] [-runinit]

[+sdf_iopath_to_prim_ok] [-sdfallowvlogescapeport][-sdfannotatepercentage]
[+sdf_nocheck_celltype] [-sdfmin | -sdftyp | -sdfmax[@<delayScale>]
[<instance>=<sdf_filename>] [-sdfmaxerrors <n>] [-sdfnoerror] [-sdfnowarn] [-
sdfreport=<filename>] [+sdf_report_unannotated_insts] [+sdf_verbose] [-
showlibsearchpath] [-stackcheck] [-std_input <filename>] [-std_output <filename>]
[+show_cancelled_e] [-stats [=+ | -]<feature>[,[+ | -]<mode>]] [-strictvital] [-suppress
{<msgNumber> | <msgGroup>} [, <msg_number> | <msgGroup>],...]] [-sv_lib
<shared_obj>] [-sv_liblist <filename>] [-sv_root <dirname>] [-sync] [-syncio | -nosyncio]

[-t [<multiplier>]<time_unit>] [-tab <tabfile>] [-tag <string>] [-title <title>] [-trace_foreign
<int>] [+transport_int_delays] [+transport_path_delays]

[-undefsyms={<args>}] [-uvmcontrol={<args>}]

[-v2k_int_delays]-vcd=[+<option>[+<option>+...]] [-vcdstim [<instance>=<filename>] [-
verboseprofile][-version] [-vhdlmergepdupackage] [-vhdlseparatepdupackage] [-
```

```
vhdl_seq_nba] [-vhdlvariablelogging] [-view [<alias_name>=<WLF_filename>] [-visual <visual>] [-vital2.2b]  
[-warning <msg_number>[,<msg_number>,...]] [-warning error] [-wlf <file_name>] [-wlfcachesize <n>] [-wlfcollapsesetdelta] [-wlfcollapsesettime] [-nowlfcollapse] [-wlfcollapse] [-nowlfcollapse] [-wlfsimcachesize <n>] [-wlfslim <size>] [-wlftlim <duration>] [-work <pathname>] [-wrealdefaultzero]
```

Description

You can simulate a VHDL configuration or an entity/architecture pair, a Verilog module or configuration.

If you specify a VHDL configuration, it is invalid to specify an architecture. During elaboration, ModelSim determines if the source has been modified since the last compile.

You can use this command in batch mode from the Windows command prompt. Refer to “[Batch Mode Simulation](#)” in the User’s Manual for more information on the VSIM batch mode.

To manually interrupt design loading, use the Break key or press <Ctrl-C> from a shell.

You can invoke vsim from a command prompt, from the Transcript window of the Main window, or from the GUI (select Simulate > Start Simulation).

Package names can be used at the command line and are treated as top-level design units.

All arguments to the vsim command are case-sensitive; for example, -g and -G are not equivalent.

Arguments

All languages

- `-allowcheckpointcpp 1|0`

(optional) Enable/disable support for checkpointing foreign C++ libraries. Must be used in the same vsim session in which the checkpoint is created. Valid arguments are 1 and 0.

1 : turn on the support

0 : turn off the support

This argument is not supported on Windows platforms.

- `-appendlog`

(optional) Append simulation data to the default log file, or the log file created with the `-logfile` argument.

- `-assertfile <filename>`

(optional) Designates an alternative file for recording VHDL assertion messages.

You can also specify an alternate file with the AssertFile *modelsim.ini* variable. By default, assertion messages are output to the file specified by the TranscriptFile *modelsim.ini* variable. Refer to [AssertFile](#), [TranscriptFile](#), and “[Creating a Transcript File](#)” in the User’s Manual for more information.

- **-autofindloop**

Attempts to find a zero delay loop and define it in a textual report when a simulation exceeds the iteration limit. The report identifies active processes along with event and signal activity, when possible. Reported results can vary, depending on design content, optimization level, and design visibility. Enabled by default, but can be disabled with `-noautofindloop`. For nonGLS designs, add `vopt +acc=l` to enhance the zero delay loop report.

- **-batch**

(optional) Runs scripted batch simulations using the `-do` argument to `vsim`. Must be specified from a Windows command prompt or a UNIX terminal. The simulator issues a warning if you use `-batch` with the `-c`, the `-gui`, or the `-i` argument to `vsim`. You can edit the BatchMode *modelsim.ini* variable to automatically run in batch mode when `-c`, `-gui`, or `-i` are not used. Refer to [BatchMode](#) in the User’s Manual.

By default, `vsim -batch` prevents automatic creation of a transcript file by disabling the TranscriptFile *modelsim.ini* variable and sending transcript data to `stdout`. You can create a transcript file by specifying the `-logfile <filename>` argument to `vsim`, or by uncommenting the BatchTranscriptFile *modelsim.ini* variable (refer to [TranscriptFile](#) and [BatchTranscriptFile](#) in the User’s Manual.) You can also disable sending transcript data to `stdout` by specifying `vsim -nostdout`, but you must then save transcript data to a file. Refer to “[Batch Mode Simulation](#)” in the User’s Manual for more information about saving transcript data.

- **+bitblast=[iopath | tcheck]**

(optional) Enables bit-blasting of specify block iopaths and timing checks (tchecks) with wide atomic ports. Without the optional qualifiers, this argument operates on both specify paths and tchecks. The qualifiers work as follows:

+bitblast=iopath — bit-blasts only specify paths with wide ports.

+bitblast=tcheck — bit-blasts only tchecks with wide ports.

This argument is intended for use with applications employing SDF annotation.

- **-c**

(optional) Specifies to run the simulator in command-line mode. Refer to “[General Modes of Operation](#)” in the User’s Manual for more information.

- **-capacity[=line]**

(optional) Enables the fine-grain analysis display of memory capacity. (The default is a coarse-grain analysis display.) The “`=line`” option allows generation of the point of allocation along with the point of declaration.

- **-capstats[=<args>[,]]**

(optional) Generates the capstats report. The possible args are:

du [+summary | +details | +duname=<duname>] | decl | line | eor | filename=<filename>

Specifying no arguments generates a summary report at the end of the simulation.

To generate a more detailed report for specific types, specify a single argument or a comma-separated list of arguments.

- du — generates a report that includes the memory usage of each design unit. You must also specify the -capacity argument with vsim.
 - The +summary argument provides a summary report of the memory usage within a design unit.
 - The +details argument provides a more expansive report that includes details about dynamic allocations.
 - The +duname=<duname> argument generates a report that covers only the specified design unit, and you can enter the argument multiple times to specify multiple design units.
- decl — generates a declaration-based report for SV dynamic objects. You must also specify the -capacity argument with vsim.
- The line argument generates a line-based report for SV dynamic objects. You must also specify the -capacity=line argument with vsim.
- eor — generates a summary report at the end of each run command.

filename=<filename> — prints the report to the specified file. If you do not specify a filename, the report prints in the transcript file.

- **-collapseall**

(optional) Enables additional optimizations on signals with user-defined resolution functions. Use the -collapseall argument only when the resolution function is associative and commutative.

- **-colormap new**

(optional) Specifies that the window have a new private colormap, instead of using the default colormap for the screen.

- **-default_radix <radix>**

(optional) Sets the default radix for the simulation and overrides the DefaultRadix preference variable. <radix> can be any of the following: ascii, binary, decimal, hexadecimal, octal, symbolic, unsigned. Refer to [DefaultRadix](#) in the User's Manual for more information.

- **-defaultstdlogicinittoz**

(optional) Sets the default VHDL initialization of std_logic to "Z" (high impedance) for ports of type OUT and INOUT. IEEE Std 1076-1987 VHDL Language Reference Manual (LRM) compliant behavior is for std_logic to initialize to "U" (uninitialized), which is incompatible with the behavior expected by synthesis and hardware.

- **-display <display_spec>**

(optional) Specifies the name of the display to use. Does not apply to Windows platforms.

For example:

```
-display :0
```

- **-displaymsgmode both | tran | wlf**

(optional) Controls the transcription of \$display system task messages to the transcript and/or the Message Viewer. Refer to “[Message Viewer Window](#)” in the User’s Manual for more information.

both — outputs messages to both the transcript and the WLF file.

tran — outputs messages only to the transcript; messages are not available in the Message Viewer. Default behavior.

wlf — outputs messages only to the WLF file/Message Viewer; messages are not available in the transcript.

The 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.

- **-do “<command_string>” | <do_file_name>**

(optional) Instructs vsim to use the command(s) specified by <command_string> or the DO file named by <do_file_name>, rather than the startup file specified in the .ini file, if any. You can specify multiple commands as a semi-colon (;) separated list. You can also specify multiple instances of -do “<command_string>” on the same command line. The commands are joined together in the order specified.

For example:

```
vsim -do "force clk 0 0, 1 10 -r 20" top -wlf top.wlf /  
-do "testfile.do" -do "run -all"
```

will become the following script:

```
"force clk 0 0, 1 10 -r 20; do testfile.do; run -all"
```

You can include nested vsim-do operations. A vsim command do-file that contains another vsim command with its own do-file executes the nested do-file.

- **-donotcollapsepartiallydriven**
(optional) Prevents the collapse of partially driven and undriven output ports during optimization. Prevents incorrect values that can occur if the output ports collapse.
- **+dumports+collapse | +dumports+nocollapse**
(optional) Determines whether vectors (VCD id entries) in dumports output are collapsed or not. The default behavior is collapsed, but you can change the default behavior by setting the DumportsCollapse variable in the *modelsim.ini* file. Refer to [DumportsCollapse](#) in the User's Manual.
- **+dumports+direction**
(optional) Modifies the format of extended VCD files to contain direction information.
- **+dumports+no_strength_range**
(optional) Ignores strength ranges when resolving driver values for an extended VCD file. This argument is an extension to the IEEE 1364 specification. Refer to [“Resolving Values”](#) in the User's Manual for additional information.
- **+dumports+unique**
(optional) Generates unique VCD variable names for ports in a VCD file, even if those ports connect to the same collapsed net.
- **-enumfirstinit**
(optional) Initializes enum variables in SystemVerilog, using the leftmost value as the default. You must also use the argument with the vlog command in order to implement this initialization behavior. Specify the EnumBaseInit variable as 0 in the *modelsim.ini* file to set this as a permanent default.
- **-error <msg_number>[,<msg_number>,...]**
(optional) Changes the severity level of the specified message(s) to error. Edit the error variable in the *modelsim.ini* file to set a permanent default. Refer to [“Message Severity Level”](#) in the User's Manual for more information.
- **-f <filename>**
(optional) Specifies a file with more vsim command arguments. Enables you to reuse complex argument strings without retyping.
Refer to [“Argument Files”](#) on page 28 for more information.
- **-fatal <msg_number>[,<msg_number>,...]**
(optional) Changes the severity level of the specified message(s) to fatal. Edit the fatal variable in the *modelsim.ini* file to set a permanent default. Refer to [“Message Severity Level”](#) in the User's Manual for more information.
- **-fsmdebug**
(optional) Enables visualization of FSMs in the GUI. You must specify this argument to view FSM information the GUI.

- **-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 from defparams (such as top-level generics/parameters and generics/parameters that would otherwise receive their default values).

You can enter multiple -g arguments, one for each generic/parameter, 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. You can prefix the name with a relative or an 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 a value for the declared data type of a VHDL generic, or any legal value for a Verilog parameter. Any value you specify for a VHDL generic must be 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 argument 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, you cannot set generics/parameters of composite type (arrays and records) 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, add 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 in escaped quotes (\\"), for example:

```
-gstrgen=\\"This is a string\\\"
```

- **-G<Name>=<Value> ...**

(optional) Same as -g (see above) except that -G also overrides generics/parameters that received explicit values in generic maps, instantiations, or from defparams.

This argument provides 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. You can prefix the name 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. The value you specify for a VHDL generic must be 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 <shared_obj>[,<shared_obj>]**

(optional) Open the specified shared object(s) with global symbol visibility. Essentially all data and functions are exported from the specified shared object and are available for other shared objects to reference and use. If you specify multiple, comma-separated, shared objects, they are merged internally and then loaded as a single shared object.

Enclose the arguments within quotation marks ("") if the shared object filenames contain any special characters or you use a space after the comma delimiter, for example:

```
-gblso "lib1.so, lib2.so"
```

You can also specify this argument with the [GlobalSharedObjectList](#) variable in the *modelsim.ini* file.

- **-gblsoauto**

(optional) Enables global symbol visibility when loading the vsim_auto_compile shared library.

- **-geometry <geometry_spec>**

(optional) Specifies the size and location of the main window. Where <geometry_spec> is of the form:

WxH+X+Y

- **-glsnegtchk [level0 | level1 | level2 | level3]**

Defines which negative timing check solver to use.

level0

Disables the solver, resulting in the simulator ignoring all negative timing checks.

level1

Enables the legacy solver, where the simulator calculates the same value of rise and fall delays.

level2

(default) Enables the LRM-compliant solver, where the simulator calculates difference values of rise and fall delays.

level3

Enables the conditional delay net solver, where the simulator calculates rise and fall delays for each condition. In cases where several conditions are true, the simulator issues a warning and the smallest delay is chosen.

- **-glspath simucondc**

Minimize the use of zero-delay output schedules of gate-level cells without requiring the use of the ifnone construct.

At the end of path selection, if the simulator does not find an active path and simultaneous input changes have occurred, this argument/value pair initiates the re-evaluation of the specify path conditions, this time ignoring the values of the inputs that have changed at this time. Specifically, the input values become don't cares for the purposes of the conditional evaluation. If the simulator selects multiple active paths, it will schedule the path with the smallest delay.

The simulator generates a suppressible warning message (vsim-17424) when this second evaluation does not select a path delay and the output schedule is made in zero-delay.

The simulator issues a warning (vsim-16072) and stops don't care evaluation if a module contains complex conditional operators, including concatenation, replication or ternary.

See [Re-evaluation of Zero-Delay Output Schedules](#) for more information.

- **-gui**

(optional) Starts the ModelSim GUI without loading a design and redirects the standard output (stdout) to the GUI Transcript window.

- **-help [all | category | <option> | <command-line> | <category_list>]**

(optional) Provides online command help.

all — List all categories and options.

category — List all command categories.

<option> — Show help for the listed command option.

<command-line> — List all options for a particular category. Vsim categories are:

- General
- Coverage
- Cpp
- Debug

- GLS
 - Language
 - Library
 - License
 - Messages
 - Multicore
 - PA
 - PDU
 - Report
 - Solver
 - Visualizer
- -i
 - (optional) Specifies that the simulator be run in interactive mode.
 - +initregNBA | +noinitregNBA
 - (optional) Controls whether +initreg settings applied to registers of sequential UDPs are non-blocking. This is useful when continuous assignments overwrite register initialization.
 - +initregNBA — (default) enables this functionality.
 - +noinitregNBA — disables this functionality.
 - -installcolormap
 - (optional) For Linux only. Instructs vsim to use its own colormap for the display.
 - -iterationlimit[=<limit>[k][m][g]] | [i[ncrease]]
 - (optional) Sets the iteration limit; or, enables a stepwise increase of the iteration limit as needed. The iteration limit specifies the number of simulation kernel iterations to allow before advancing time. The default iteration limit is 10 million, set by the IterationLimit variable of the *modelsim.ini* file. Refer to [IterationLimit](#) in the User's Manual.
 - =<limit>[k][m][g]
 - <limit> – An unsigned integer value that overrides the iteration limit set by the IterationLimit variable. Use the “set IterationLimit <value>” command to override this value.
 - k – Designates units in thousands.
 - m – Designates units in millions.
 - g – Designates units in billions.
 - =i[ncrease]

Sets the simulator to automatically increase the iteration limit—if the default iteration limit of 10 million is reached—and report the final setting at the end of simulation. If the simulator reaches the default limit of 10 million iterations, it issues a warning about a possible zero delay loop, and sets a flag to report the final iteration limit at the end of simulation. Each time the iteration exceeds the new limit, the value is increased by 500 thousand. The value that the simulator reports at the end of simulation is the final iteration limit that was set, not the maximum simulator iterations that were performed. The maximum allowed iteration limit value 4,294,967,000.

- **-keeploaded**

(optional) Prevents the simulator from unloading/reloading any HDL interface shared libraries when it restarts or loads a new design. The shared libraries remain loaded at their current positions. User application code in the shared libraries must reset its internal state during a restart for this to work effectively.

- **-keeploadedrestart**

(optional) Prevents the simulator from unloading/reloading any HDL interface shared libraries during a restart. The shared libraries remain loaded at their current positions. User application code in the shared libraries must reset its internal state during a restart for this to work effectively.

Use this argument if you will be doing warm restores after a restart, and the user application code has set callbacks in the simulator. If you do not use this argument, and the shared library is loaded into a new location, the callback function pointers might not be valid.

- **-keepstdout**

(optional) For use with foreign programs. Instructs the simulator not to redirect the stdout stream to the Main window.

- **-logfile <filename> | -l <filename> | -nolog**

(optional) Controls saving of transcript data during batch and regular simulations.

-logfile <filename> — Saves transcript data to <filename>. You can use the abbreviation **-l <filename>**. Overrides the default transcript file creation set with the TranscriptFile or BatchTranscriptFile *modelsim.ini* variables. You can also specify “stdout” or “stderr” for <filename>. Refer to [TranscriptFile](#) or [BatchTranscriptFile](#) in the User’s Manual.

-nolog — Disables transcript file creation. Overrides the TranscriptFile or BatchTranscriptFile variables set in the *modelsim.ini* file.

Use **-appendlog** to append simulation data to the log file.

Refer to “[Batch Mode](#)” in the User’s Manual for more information about saving transcript data.

- **-lib <libname>**

(optional) Specifies the working library in which vsim will look for the design unit(s). Default is "work".

- **-libverbose[=prlib]**

(optional) Enables verbose messaging about library search and resolution operations. The “=prlib” option prints out the -L or -Lf argument that was used to locate each design unit loaded by vsim. This information is printed to the right of the existing “Loading design unit xyz . . .” messages.

Libraries containing top design units that are not explicitly present in the set of -L/-Lf arguments are implicitly promoted to searchable libraries at the end of the library search order. They appear as -Ltop in the output of the -libverbose argument. To stop creation of -Ltop libraries, use the -noltop argument of vsim.

- **-L <library_name> ...**

(optional) Specifies the library to search for top level design units instantiated from Verilog, and for VHDL default component binding. Prints a list of all visible top level libraries, if a top level design unit cannot be found. Refer to “[Library Usage](#)” in the User’s Manual for more information. If you specify multiple libraries, each must be preceded by the -L argument. Libraries are searched in the order in which they appear on the command line.

- **-Ldir <pathname> [<pathname> ...]**

(optional) Passes one or more container folders for libraries specified by either vsim -L or vsim -Lf. Once you specify a container folder (pathname), you can directly reference the libraries in this folder using their logical names. You can specify multiple values for pathname. ModelSim searches multiple paths in the order in which you specify them on the command line.

Note

 ModelSim searches the current working directory (\$cwd) before any pathnames you specify for -Ldir, as if there was an implicit vsim -Ldir . specified first on the command line.

- **-Lf <library_name> ...**

(optional) Same as -L but libraries are searched before `uselib directives. Refer to “[Library Usage](#)” in the User’s Manual for more information.

- **-modelsimini <path/modelsim.ini>**

(optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified for the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file, including the filename. On Windows systems, use a forward slash (/) for the path separator character.

Restriction

 Do not use this argument in a file specified with the vsim -f argument. ModelSim will display an error message.

- **-msgfile <filename>**

(optional) Designates an alternative file for recording error messages. You can also specify an alternate file with the Error File *modelsim.ini* variable. By default, error messages are output to the file specified by the *TranscriptFile* variable in the *modelsim.ini* file. Refer to [ErrorFile](#) and [TranscriptFile](#) in the User's Manual.

- **-msglimit {error | warning | all[,-<msgNumber>,...] | none[,+<msgNumber>,...] | <msgNumber>,[msgNumber,...]}**

(optional) Limits messages to the default message limit count of five. Use the **-msglimitcount** argument to change the default message limit count. Specify multiple messages as a comma-separated list.

error — Stops the run when the total number of errors reaches the default count.

warning — Stops the run when the total number of warnings reaches the default count.

all — Limits all messages to the default count except those you specifically exclude. To exclude messages from the limit, supply a comma-separated list of message numbers, each preceded by a minus sign (-), for example “all,-<msgNumber1>,-<msgNumber2>,...”.

none — Excludes all messages from the default count except those you specifically include. To include messages to limit to the default count, supply a comma-separated list of message numbers, each preceded by a plus sign (+), for example “none,+<msgNumber1>,+<msgNumber2>,...”.

<msgNumber>[,<msgNumber>,...] — Specifies messages to limit to the default count.

Examples:

- Stop the run when the total number of errors reaches the default count:

```
-msglimit error
```

- Stop the run when the total number of warnings reaches the default count:

```
-msglimit warning
```

- Specifically limit messages 2374 and 2385 to the default count:

```
-msglimit 2374,2385
```

- Limit all messages to the default count, except messages 2374 and 2385:

```
-msglimit all,-2374,-2385
```

- Limit only messages 2374 and 2385 to the default count:

```
-msglimit none,+2374,+2385
```

- **-msglimitcount <limit_value>** -msglimit [all, | none,] <msgNumber>[, <msgNumber>...]
(optional) Limits the reporting of listed messages to user-defined limit_value. Overrides the MsgLimitCount variable in the *modelsim.ini* file. Refer to [MsgLimitCount](#) in the User's Manual.
- **-msglimitcount <limit_value>** -msglimit [error|warning]
(optional) Stops the run when the total number of reported errors/warnings reaches the user-defined limit_value.
- **-msgmode both | tran | wlf**
(optional) Specifies the location(s) for the simulator to output elaboration and runtime messages.
 - both — Outputs messages to both the transcript and the WLF file.
 - tran — (default) 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.

Refer to "[Message Viewer Window](#)" in the User's Manual for more information.

- **-msgsingleline**
(optional) Displays each message in a single line.
- **-multisource_delay min | max | latest**
(optional) Controls the handling of multiple PORT or INTERCONNECT constructs that terminate at the same port. By default, the Module Input Port Delay (MIPD) is set to the max value encountered in the SDF file. Alternatively, you can 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**
(optional) Enables multisource interconnect delay with pulse handling and transport delay behavior. Works for both Verilog and VITAL cells.

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 arguments (described below).

You cannot use the +multisource_int_delays argument 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>**
(optional) Specifies the application name used by the interpreter for send commands. This does not affect the title of the window.

- **-noautofindloop**
(optional) Disables `-autofindloop`.
- **-noautoldlibpath**
(optional) Disables the default internal setting of `LD_LIBRARY_PATH`, enabling you to set it yourself. Use this argument to make sure that `LD_LIBRARY_PATH` is not set automatically while you are using the GUI,
- **-nocapacity**
(optional) Disables the display of both coarse-grain and fine-grain analysis of memory capacity.
- **-nocompress**
(optional) Causes VSIM to create uncompressed checkpoint files. You can also specify this argument with the `CheckpointCompressMode` variable in the *modelsim.ini* file. Refer to [CheckpointCompressMode](#) in the User's Manual.
- **-noimmedca**
(optional) 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.” You can also set even ordering with the `ImmediateContinuousAssign` variable in the *modelsim.ini* file. Refer to [ImmediateContinuousAssign](#) in the User's Manual.
- **+no_notifier**
(optional) Disables the toggling of the notifier register argument of all timing check system tasks. By default, the notifier toggles 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.
- **+nospecify**
(optional) Disables specify path delays and timing checks in Verilog.
- **-nostdout**
(optional) Directs all output to the transcript only when in command line and batch mode. Prevents duplication of I/O between the shell and the transcript file. Has no affect on interactive GUI mode. Refer to “[Batch Mode Simulation](#)” in the User's Manual for information about batch mode usage.
- **+no_tchk_msg**
(optional) Disables error messages generated when timing checks are violated. 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.

- **-note <msg_number>[,<msg_number>,...]**
(optional) Changes the severity level of the specified message(s) to "note." Edit the note variable in the *modelsim.ini* file to set a permanent default. Refer to [note](#) and "[Message Severity Level](#)" in the User's Manual for more information.
- **+notifier_ondetect**
(optional) Causes X output values generated by timing check notifier toggles to be scheduled with zero delay.
- **+notimingchecks | +ntcnotchks**
(optional) Removes Verilog timing checks and disables VITAL timing checks for all instances in the specified design (This option sets the generic TimingChecksOn to FALSE for all VHDL Vital models with the Vital_level0 or Vital_level1 attribute. Does not affect generics with the name TimingChecksOn on non-VITAL models.) By default, Verilog timing check system tasks (\$setup, \$hold,...) in specify blocks are enabled. For VITAL, the ASIC or FPGA vendor controls the timing check default, but most default to enabled.

If you are using the three-step flow, pass +notimingchecks to vopt instead of vsim.

Additionally, +ntcnotchks maintains the delay net delays that negative timing check limits make necessary. For this reason when using +ntcnotchks it is necessary to SDF annotate all timing check values.
- **-nowiremodelforce**
(optional) Restores the [force](#) command to the previous usage model (prior to version 10.0b), where an input port cannot be forced directly if it is mapped at a higher level in VHDL and mixed models. Signals must be forced at the top of the hierarchy connected to the input port.
- **-optionset <optionset_name>**
(optional) Calls an optionset as defined in the *modelsim.ini* file. Refer to "[Optionsets](#)" on page 27 in the Reference Manual for more information.
- **-pduignore[=<instpath>]**
Ignore Preoptimized Design Unit (black-box). If you do not specify <instpath>, all PDUs found in compiled libraries are ignored. Otherwise, the PDU specified by <instpath> is ignored. You can specify this argument multiple times using different values of <instpath>. Equivalent to the deprecated -ignore_bbox argument.
- **-pdupath[=<lib_path>]**
(optional) Specifies library path for a preoptimized design unit (PDU) when the library is moved after top level optimized design unit creation. If you do not specify <lib_path>, the library path is the current working directory.
- **-permissive**
(optional) Allows messages in the LRM group of error messages to be downgraded to a warning.

You can produce a complete list of error messages by entering the following command:

```
verror -kind vsim -permissive
```

- **-postsimdataflow**
(optional) Makes Dataflow window available for post simulation debug operations. By default, the Dataflow window is not available for post-sim debug.
- **-printsimstats[=[<val>][v]]**
(optional) Prints the output of the [simstats](#) command to the transcript at the end of simulation, before exiting. <val> is 0 - disables simstats, 1(default) - prints stats at the end of simulation,2 - prints out stats at the end of each run command and simulation. v- prints out verbose statistics, including the checkout time.
Each performance statistic is printed with its related units on a separate line. Edit the PrintSimStats variable in the *modelsim.ini* file to set the simulation to print the simstats data by default. Refer to [PrintSimStats](#) in the User's Manual.
The command `vsim -printsimstats=v` is equivalent to `vsim -stats=perf+verbose`. The command `vsim -printsimstats=2v` is equivalent to `vsim -stats=perf+verbose+eor`.
- **+pulse_int_e/<percent>**
(optional) 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. 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 you do not specify the rejection limit, 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>**
(optional) 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. This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source can be VITAL or Verilog.
A pulse less than the rejection limit is filtered. If you do not specify the error limit with `+pulse_int_e`, it defaults to the rejection limit.
- **-quiet**
(optional) Disables 'Loading' messages during batch-mode simulation.

- `+questa_mvc_core+wlf_disable <filename>`
(optional) Disables the creation of a WLF file.
- `-rngtrace[=<filename>]`
(optional) Enables Random Number Generator (RNG) trace reporting feature.
`<filename>` — Directs the RNG trace related output to the specified file.
- `-rngtraceopt=[+|-]<option_string>[,[+|-]<option_string>]`
(optional) Specifies configuration options for the RNG trace generated output.
+ — Enables the `<option_string>`
- — Disables the `<option_string>`

If you do not specify either a “+” or “-”, the command assumes you are enabling the specified `<option_string>`.

Following are the valid values of `<option_string>`. By default, the options are disabled.

`showciid` — Displays the ciid of class instances.

`showsimeTime` — Displays the simulation time.

`useprocessid` — Displays the process names based on the instance ID of the associated process class. With this option, the process names are formatted as the following:

`#<kind>#<ciid>#`

where `<kind>` is INITIAL, ALWAYS, FORK, and so on, and `<ciid>` is the class instance ID of the process class object.

- `-rngtraceclassfilter=<class_filter_string>[,<class_filter_string>]`
(optional) Restricts the RNG trace generated output for class instance changes to the class instances that match one of the specified `<class_filter_string>` strings.

Note

 A `<class_filter_string>` of `TFoo` matches the class types `TFoo`, `TFoo_1`, `TFooBar`, and so on. To match all types of a parameterized class `TBar`, specify a prefix, `TBar_`.

- `-rngtraceprocessfilter=<scope_filter_string>[,<scope_filter_string>]`
(optional) Restricts the RNG trace generated output for process and class instance changes to the processes that match one of the specified scopes.
- `-runinit`
(optional) 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`
(optional) 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 when a corresponding specify path is not found. Refer to “[SDF to Verilog Construct Matching](#)” in the User’s Manual for additional information.

- **-sdfallowvlogescapeport**

Enables SystemVerilog language extension related to SDF annotation. Matches port name in RTL (“`<portname>[0]`”) with either of the following in SDF file:

- non-escaped port name (“`<portname>[0]`”)
- escaped port name (“`\\<portname>[0]`”)

Allows for matching bit-blasted scalar port names in RTL with normal non-escaped port names in SDF.

- **-sdfannotatepercentage**

(optional) Produces a report in the transcript containing annotation percentage for path delays and timing checks defined in specify blocks.

- **-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. Can also specify instances under VHDL generates as the SDF back-annotation point.

@<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.

<instance>= — A specific instance for the associated SDF file. Use this when not performing backannotation at the top level.

<sdf_filename> — The file containing the SDF information.

- **-sdfmaxerrors <n>**

(optional) Controls the number of Verilog SDF missing instance messages to generate before terminating vsim. **<n>** is the maximum number of missing instance error messages. The default number is 5.

- **-sdfnoerror**

(optional) Changes SDF errors to warnings so that the simulation can continue. Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not.

- **-sdfnowarn**

(optional) Disables warnings from the SDF reader. Refer to “[VHDL Simulation](#)” in the User’s Manual for an additional discussion of SDF.

- **-sdfreport=<filename>**

(optional) Produces a report at the location of **<filename>** containing information about unannotated and partially-annotated specify path objects, specifically path delays and

timing checks. Refer to “[Reporting Unannotated Specify Path Objects](#)” in the User’s Manual for more information.

- `+sdf_report_unannotated_insts`

(optional) 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`

(optional) 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. When you also specify the `+multisource_int_delays` argument, the `+sdf_verbose` argument adds more detail to the output of the [write timing](#) command.

- `-stackcheck`

(optional) Enables runtime stack usage sanity checking. This argument enables vsim to add additional instrumentation at runtime to monitor the system stack usage. If the usage exceeds the reserved stack limit, vsim will report a fatal error. An uncaught stack overflow can lead to a random downstream crash.

- `-stats [=+ | -]<feature>[,[+ | -]<mode>]`

(optional) Controls display of statistics sent to a logfile, stdout, or the transcript. Specifying `-stats` without options sets the default features (cmd, msg, and time).

Specify multiple features and modes for each instance of `-stats` as a comma-separated list. You can specify `-stats` multiple times on the command line, but only the final instance takes effect.

You can specify `-printsimstats` and `-stats` on the same command line, however `-stats` always overrides `-printsimstats`, regardless of the order in which you specify the options.

`[+ | -]` — Controls activation of the feature or mode. The plus character (+) enables the feature, and the minus character (-) disables the feature. You can also enable a feature or mode by specifying it without the plus (+) character. Setting this argument adds or subtracts features and modes from the default settings "cmd,msg,time".

`<feature>`

`all` — Display all statistics features (cmd, msg, perf, time). Mutually exclusive with `none` option. When specified in a string with other options, `all` is applied first.

`cmd` — (default) Echo the command line.

`msg` — (default) Display error and warning summary at the end of command execution.

`none` — Disable all statistics features. Mutually exclusive with `all` option. When specified in a string with other options, `none` is applied first.

`perf` — Display time and memory performance statistics.

`time` — (default) Display Start, End, and Elapsed times.

Note

 Use -stats=perf+verbose to display information about the operating system and host machine.

<mode>

You can set modes for a specific feature, or globally for all features. To add or subtract a mode for a specific feature, use the plus (+) or minus (-) character with the feature, for example, vcover dump -stats=cmd+verbose,perf+list. To add or subtract a mode globally for all features, specify the modes in a comma-separated list, for example, vcover dump -stats=time,perf,list,-verbose. You cannot specify global and feature specific modes together.

eor — Print performance statistics at the end of each run command. Valid for use only with perf feature (-stats=perf+eor); it is invalid with other features.

kb — Print statistics in kilobyte units with no auto-scaling.

list — Display statistics in a Tcl list format when available.

verbose — Display verbose statistics information when available.

Note

 By default, vsim prints the command line with the '-f filename' option. Prior to 10.3c, behavior was to print the command line with expanded arguments from '-f filename'. To enable the previous behavior, specify -stats=cmd+verbose.

Refer to [Tool Statistics Messages](#) in the User's Manual for more information.

- -suppress {<msgNumber> | <msgGroup>} {[,<msg_number> | <msgGroup>] ,...]
(optional) Prevents the specified message(s) from displaying. You cannot suppress Fatal or Internal messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to [suppress](#) and “[Message Severity Level](#)” in the User's Manual for more information.

<msgNumber>

A specific message number

<msgGroup>

A value identifying a predefined group of messages, based on area of functionality. These groups suppress only notes or warnings. The valid arguments are:

All, GroupNote, GroupWarning, GroupFLI, GroupPLI, GroupSDF, GroupVCD, GroupVital, GroupWLF, GroupTCHK, GroupPA, GroupLRM

- -sync
(optional) Executes all X server commands synchronously, so that errors are reported immediately. Does not apply to Windows platforms.
- -syncio | -nosyncio
(optional) Controls buffering of text output to console and logfile.

- syncio — (default) I/O is synchronous with simulation activity, always up-to-date.
- nosyncio — Disables I/O synchronization. This enables vsim to run faster by buffering (delaying) output.

- **-t [<multiplier>]<time_unit>**

(optional) Specifies the simulator time resolution. <time_unit> must be one of the following:

fs, ps, ns, us, ms, sec

The default is 1ps; the optional <multiplier> can be 1, 10 or 100.

Do not put a 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:

- VHDL design — Uses the value specified for the Resolution variable in *modelsim.ini*.
- Verilog design with ‘timescale directives — Uses the minimum specified time precision of all directives.
- Verilog design with no ‘timescale directives — Uses the value specified for the Resolution variable in the *modelsim.ini* file. Refer to [Resolution](#) in the User’s Manual.
- Mixed design with VHDL on top — Uses the value specified for the Resolution variable in the *modelsim.ini* file.
- Mixed design with Verilog on top:
 - For Verilog modules not under a VHDL instance — Uses the minimum value specified for their ‘timescale directives.
 - For Verilog modules under a VHDL instance — Ignores all ‘timescale directives (uses the minimum value for ‘timescale directives in all modules not under a VHDL instance).

If there are no ‘timescale directives in the design, uses the value specified for the Resolution variable in *modelsim.ini*.

Tip

 After you have started a simulation, you can view the current simulator resolution by entering [report](#) simulator state.

- **-tab <tabfile>**

(optional) 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.

- **-tag <string>**
(optional) Specifies a string tag to append to foreign trace filenames. Used with the **-trace_foreign <int>** option. Used when running multiple traces in the same directory.
- **-title <title>**
(optional) Specifies the title to appear for the ModelSim Main window. If you omit this argument, the window title is the current ModelSim version. Useful when running multiple simultaneous simulations. Text strings with spaces must be in quotes (for example, "my title").
- **-trace_foreign <int>**
(optional) 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 the actions of the foreign interface side.

The purpose of the logfile is to aid the debugging of your 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 PLI/VPI code.

- **-undefsyms={<args>}**
(optional) Manages the undefined symbols in the shared libraries currently being loaded into the simulator. You can also manage undefined symbols with the UdefSyms *modelsim.ini* variable. Refer to [UdefSyms](#) in the User's Manual.
 {<args>}

You must specify at least one argument.

on — (default) Enables automatic generation of stub definitions for undefined symbols, and permits loading of the shared libraries despite the undefined symbols.

off — Disables loading of undefined symbols. Undefined symbols trigger an immediate shared library loading failure.

verbose — Permits loading to the shared libraries, despite the undefined symbols, and reports the undefined symbols for each shared library.

- **-uvmcontrol={<args>}**
(optional) Controls UVM-Aware debug features. These features work with either a standard Accelera-released open source toolkit or the pre-compiled UVM library package in ModelSim.

 {<args>}

You must specify at least one argument. You can enable or disable some arguments by prefixing the argument with a dash (-). Refer to the argument descriptions for more information.

- all — Enables all UVM-Aware functionality and debug options except disable and verbose. You must specify verbose separately.
- certe — Enables the integration of the elaborated design in the Certe tool. Disables Certe features when specified as -certe.
- disable — Prevents the UVM-Aware debug package from being loaded. Changes the results of randomized values in the simulator.
- msglog — Enables messages logged in UVM to be integrated into the Message Viewer. You must also enable wlf message logging by specifying tran or wlf with vsim -msgmode. Disables message logging when specified as -msglog
- none — Turns off all UVM-Aware debug features. Useful when multiple -uvmcontrol options are specified in a separate script, makefile or alias and you want to be sure all UVM debug features are turned off.
- reseed — Disables behavior of UVM simulation, where if you reseed the simulation, the random sequences generated by UVM will change.
- struct — (default) Enables UVM component instances to appear in the Structure window. UVM instances appear under “uvm_root” in the Structure window. Disables Structure window support when specified as -struct.
- trlog — Enables or disables UVM transaction logging. Logs UVM transactions for viewing in the Wave window. Disables transaction logging when specified as -trlog.
- verbose — Sends UVM debug package information to the transcript. Does not affect functionality. Must be specified separately.

You can specify arguments as multiple instances of -uvmcontrol. Specify multiple arguments as a comma-separated list, without spaces. For example,

```
vsim -uvmcontrol=all,-trlog
```

enables all UVM features except UVM transaction logging. Where arguments are in conflict, the final argument overrides earlier arguments and a warning is issued.

You can also control UVM-Aware debugging with the UVMControl modelsim.ini variable. Refer to [UVMControl](#) in the User’s Manual.

- -vcdstim [<instance>=>]<filename>
(optional) Specifies a VCD file from which to re-simulate the design.

Note

 You must have an existing VCD file to use this command. To create a VCD file, you must first execute the +dumpports+nocollapse or +dumpports+collapse options in a ModelSim simulation, then execute the [vcd dumpports](#) command. Refer to “[Using Extended VCD as Stimulus](#)” in the User’s Manual for more information.

- -verboseprofile
(optional) Enables enhanced kernel routines that simulate with additional data to improve performance profiler data collection.

- **-version**
(optional) Returns the version of the simulator as used by the licensing tools.
- **-view [<alias_name>=<WLF_filename>]**
(optional) Specifies a wave log format (WLF) file for vsim to read. Enables you to use vsim to view the results from an open simulation (*vsim.wlf*) or an earlier saved simulation. You can open the Structure, Objects, Wave, and List windows to look at the results stored in the WLF file (other ModelSim windows do 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. Allows wildcard characters.
 - <WLF_file_name> — Specifies the pathname of a saved WLF file.
- See additional discussion in the Examples.
- **-visual <visual>**
(optional) Specifies the visual to use for the window. Does not apply to Windows platforms.
Where <visual> can be:
 - <class> <depth> — One of the following:
 - {directcolor | grayscale | greyscale | pseudocolor | staticcolor | staticgray | staticgrey | truecolor}
 - followed by:
 - <depth> — Specifies the number of bits per pixel 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 the number of bits per pixel for the visual.
- **-warning <msg_number>[,<msg_number>,...]**
(optional) Changes the severity level of the specified message(s) to "warning." Edit the warning variable in the *modelsim.ini* file to set a permanent default. Refer to [warning](#) and "[Message Severity Level](#)" in the User's Manual for more information.
- **-warning error**
(optional) Reports all warnings as errors.
- **-wlf <file_name>**
(optional) Specifies the name of the wave log format (WLF) file to create. The default file name is *vsim.wlf*. You can also specify this option with the WLFFilename variable in the *modelsim.ini* file. Refer to [WLFFilename](#) in the User's Manual.

- **-wlfcachesize <n>**
(optional) Specifies the size in megabytes of the WLF reader cache. 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 can be beneficial in slow network environments. You can also specify this option with the WLFCacheSize variable in the *modelsim.ini* file. Refer to [WLFCacheSize](#) in the User's Manual.
- **-wlfcollapsedelta**
(default) Instructs ModelSim to record values in the WLF file only at the end of each simulator delta step, and ignore any sub-delta values. Can reduce WLF file size. You can also specify this option with the WLFCollapseMode variable in the *modelsim.ini* file. Refer to [WLFCollapseMode](#) in the User's Manual.
- **-wlfcollapsetime**
(optional) Instructs ModelSim to record values in the WLF file only at the end of each simulator time step, and ignore delta or sub-delta values. Can reduce WLF file size. You can also specify this option with the WLFCollapseMode variable in the *modelsim.ini* file.
- **-nowlfcollapse**
(optional) Instructs ModelSim to preserve all events for each logged signal, and their event order, to the WLF file. Can result in relatively larger WLF files. You can also specify this option with the WLFCollapseMode variable in the *modelsim.ini* file.
- **-wlfcompress**
(default) Creates compressed WLF files. Use -nowlfcompress to turn off compression. You can also specify this option with the WLFCompress variable in the *modelsim.ini* file. Refer to [WLFCompress](#) in the User's Manual.
- **-nowlfcompress**
(optional) Causes vsim to create uncompressed WLF files. The default is to compress WLF files to reduce file size. This can slow simulation speed by one to two percent. Disabling compression speeds up simulation, and can help if you are experiencing problems with faulty data in the compressed WLF file. You can also specify this option with the WLFCompress variable in the *modelsim.ini* file.
- **-wlfdelteonquit**
(optional) Deletes the current simulation WLF file (*vsim.wlf*) automatically when the simulator exits. You can also specify this option with the WLFDelteOnQuit variable in the *modelsim.ini* file. Refer to [WLFDelteOnQuit](#) in the User's Manual.
- **-nowlfdelteonquit**
(default) Preserves the current simulation WLF file (*vsim.wlf*) when the simulator exits. You can also specify this option with the WLFDelteOnQuit variable in the *modelsim.ini* file. Refer to [WLFDelteOnQuit](#) in the User's Manual.

- **-wlfllock**
(optional) Locks a WLF file. An invocation of ModelSim will not overwrite a WLF file that is being written by a different invocation.
- **-nowlfllock**
(optional) Disables WLF file locking. This prevents vsim from checking whether a WLF file is locked prior to opening it, and prevents vsim from attempting to lock a WLF once it has been opened.
- **-nowlfopt**
(optional) Disables optimization of waveform display in the Wave window. You can also specify this option with the [WLFOptimize](#) variable in the *modelsim.ini* file.
- **-wlfsimcachesize <n>**
(optional) Specifies the size in megabytes of the WLF reader cache for the current simulation dataset only. By default, the cache size is set to zero. This enables you to set one size for the WLF reader cache used during simulation, and a different size for the caches used during post-simulation debug. WLF reader caching caches blocks of the WLF file to reduce redundant file I/O. If you do not specify the -wlfsimcachesize argument or the [WLFSimCacheSize](#) *modelsim.ini* variable, the -wlfcachesize argument or the [WLFSimCacheSize](#) *modelsim.ini* variable settings are used. Refer to [WLFSimCacheSize](#) and [WLFCacheSize](#) in the User's Manual.
- **-wlfslim <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

 Note that a WLF file contains event, header, and symbol portions. The size restriction is placed on the event portion only. Consequently, the resulting file will be larger than the specified size.

If used in conjunction with -wlftlim, the more restrictive of the limits takes precedence.

You can also specify this option with the [WLFSizeLimit](#) variable in the *modelsim.ini* file. Refer to [WLFSizeLimit](#) and “[Limiting the WLF File Size](#)” in the User's Manual.

- **-wlftlim <duration>**
(optional) Specifies the duration of simulation time for WLF file recording. The default is infinite time (0). The <duration> is an integer of simulation time at the current resolution; optionally, you can specify the resolution by placing 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 -wlfslim, the more restrictive of the two limits takes effect.

You can also specify this option with the WLFTimeLimit variable in the *modelsim.ini* file. Refer to [WLFTimeLimit](#) in the User's Manual.

The -wlfslim and -wlftlim arguments are intended to limit WLF file sizes for long or heavily logged simulations. If you use small values for these arguments, they may be overridden by the internal granularity limits of the WLF file format. Refer to “[Limiting the WLF File Size](#)” in the User's Manual.

VHDL Arguments

- **-absentisempty**

(optional) Specifies to treat any VHDL files as empty that, when opened for read, target non-existent files. Keeps ModelSim from issuing fatal error messages.

- **-accessobjdebug**

(optional) Enables logging of VHDL access type variables—both the variable value and any access object that the variable points to during the simulation. Also changes the default form of display-only names (such as [10001]) to a different form that you can use as input to any command that expects an object name.

By default, logging is turned off. This means that while access variables themselves can be logged and displayed in the ModelSim display windows, any access objects that they point to will not be logged.

Overrides the setting for the AccessObjDebug variable in the *modelsim.ini* file. Refer to [AccessObjDebug](#) in the User's Manual.

- **-noaccessobjdebug**

(optional) Disables logging of VHDL access type variables, which is the default setting. This means that while access variables themselves can be logged and displayed in the ModelSim display windows, any access objects that they point to will not be logged.

Overrides the setting for the AccessObjDebug variable in the *modelsim.ini* file.

- **-nocollapse**

(optional) Disables the optimization of internal port map connections.

- **-nofileshare**

(optional) Turns off file descriptor sharing. By default ModelSim shares a file descriptor for all VHDL files opened for write or append that have identical names.

- **-noglitch**
(optional) Disables VITAL glitch generation.
Refer to “[VHDL Simulation](#)” in the User’s Manual for additional discussion of VITAL.
- **+no_glitch_msg**
(optional) Disable VITAL glitch error messages.
- **-noverboseprofile**
(optional) Disables the routines when used with -autoprofile argument.
- **-novhdlvariablelogging**
(optional) Turns off the ability to log recursively or add process variables to the Wave or List windows. Refer to -vhdlvariable logging. Refer also to [VhdlVariableLogging](#) in the User’s Manual.
- **-std_input <filename>**
(optional) Specifies the file to use for the VHDL TextIO STD_INPUT file.
- **-std_output <filename>**
(optional) Specifies the file to use for the VHDL TextIO STD_OUTPUT file.
- **-strictvital**
(optional) Specifies to exactly match the VITAL package ordering for messages and delta cycles. Useful for eliminating delta cycle differences caused by optimizations not addressed in the VITAL LRM. Using this argument negatively impacts simulator performance.
- **+transport_int_delays**
(optional) Selects transport mode with pulse control for single-source nets (one interconnect path). 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 can be VITAL or Verilog. This option works independently from +multisource_int_delays.
- **+transport_path_delays**
(optional) Selects transport mode for path delays. 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.
- **-vcd=[+<option>[+<option>+...]]**
Specifies VCD logging-related option settings. Options are:
 - multid — Enable logging for multi-dimensional arrays.
 - maxmultid=<max-limit> — Override limit on size of multi-dimensional arrays that are logged. The default size limit is 1024 elements.

- **-vhdlmergepdupackage**

(optional) Turns off sharing of one package between two PDUs. Each PDU will have a separate copy of the package. You can also specify this option with the `VhdlSeparatePduPackage` variable in the `modelsim.ini` file. Refer to [VhdlSeparatePduPackage](#) in the User's Manual.

- **-vhdlseparatepdupackage**

(optional, default) Turns on sharing of packages between two or more PDUs.

- **-vhdl_seq_nba**

Enables the scheduling of identified VHDL clocked processes in the NBA region instead of the Active Region. This is a non-LRM approach that avoids certain race conditions in mixed language designs. In contrast, the LRM signal assignment behavior executes processes in the Active Region.

Use this argument carefully, as it may result in glitches and delta differences in simulation results, and it is possible that the simulator will still execute some clocked processes in the active region, even when using `-vhdl_seq_nba`.

The simulator ignores `-vhdl_seq_nba` for pure VHDL designs.

- **-vhdlvariablelogging**

(optional) Allows process variables to be logged recursively or added to the Wave and List windows (process variables can still be logged or added to the Wave and List windows explicitly with or without this argument).

You can disable this argument with `-novhdlvariablelogging`. Refer to `-vhdlvariable logging`. Refer also to [VhdlVariableLogging](#) in the User's Manual.

Note

 Logging process variables decreases simulation performance. The recommended procedure is to not log or add process variables to the Wave and List windows. However, if debugging does require logging them, you can use this argument to minimize the performance decrease.

- **-vital2.2b**

(optional) Selects SDF mapping for VITAL 2.2b (default is VITAL 2000).

Verilog Arguments

- **+alt_path_delays**

(optional) Configures path delays to operate in inertial mode by default. In inertial mode, a pending output transition is canceled when a new output transition is scheduled. The result is that an output can 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 canceled 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 canceled pending value of the net. This option has no effect in transport mode (see `+pulse_e/<percent>` and `+pulse_r/<percent>`).

- `-checkvifacedrivers`
(optional) Turns off/on checks for multiple-driver analysis in assignments made through virtual interfaces.
- `-classdebug | -noclassdebug`
(optional) Enables/disables visibility into class instances, and includes SystemVerilog queues, dynamic arrays, and associative arrays for class and UVM debugging. You can also enable visibility into class instances by setting the ClassDebug modelsim.ini variable to 1. Refer to [ClassDebug](#) in the User's Manual. Refer also to the [classinfo find](#) command.
- `+delayed_timing_checks`
(optional) Causes timing checks to be performed on the delayed versions of input ports (used when there are negative timing check limits). By default, ModelSim automatically detects and applies `+delayed_timing_checks` to cells with negative timing checks. To turn off this feature, specify `+no_autodtc` with vsim.
- `-dpiforceheader`
(optional) Forces the generation of a DPI header file even if it will be empty of function prototypes.
- `-dpiheader`
(optional) Generates a header file that can then be included in C source code for DPI import functions. Simulation quits after generation of the header file. Refer to “[DPI Use Flow](#)” in the User's Manual for additional information.
- `-dpilib <libname>`
(optional) Specifies the design library name that contains DPI exports and automatically compiled object files. If you do not set the `-dpilib` argument, vsim loads export symbols from all libraries accessible using vsim arguments `-L`, `-Lf`, and `-lib`. Multiple occurrences of `-dpilib` are supported.
- `-dпиoutoftheblue 0 | 1 | 2`
(optional) 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.
- `-gconrun | -nogconrun`
(optional) Enables/disables garbage collector execution after each simulation [run](#) command completes.

- **-gconstep | -nogconstep**
(optional) Enables/disables garbage collector execution after each step when stepping through your simulation.
- **-gcthreshold <n>**
(optional) Sets the maximum amount of memory in megabytes allocated for storage of class objects. When the threshold is reached, the garbage collector runs to delete unreferenced objects.

<n> — Any positive integer where **<n>** is the number of megabytes. The default size is 100 megabytes.
Refer to the related [GCThreshold](#) and [GCThresholdClassDebug](#) *modelsim.ini* file variables.
- **-hazards**
(optional) Enables event order hazard checking in Verilog modules (Verilog only). You must also specify this argument when you compile your design with [vlog](#). Refer to “[Hazard Detection](#)” in the User’s Manual for more details.

Note

 Using **-hazards** implicitly enables the **-compat** argument. As a result, using this argument may affect your simulation results.

- **-ignoreinilibs**
(optional) Ignore the libraries specified in the LibrarySearchPath variable in the vsim section of the *modelsim.ini* file. Refer to [LibrarySearchPath](#) in the User’s Manual.
- **-noappendclose**
(optional) Simulator does not immediately close files opened in APPEND mode. Designed to override the AppendClose *modelsim.ini* variable when it is set to one (On). Subsequent calls to file_open in APPEND mode will therefore not require operating system interaction, resulting in faster performance. Refer to [AppendClose](#) in the User’s Manual.
- **+no_autodtc**
(optional) Turns off auto-detection of optimized cells with negative timing checks and auto-application of +delayed_timing_checks to those cells.
- **+no_cancelled_e_msg**
(optional) Disables negative pulse warning messages. 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.
- **+no_neg_tchk**
(optional) Disables negative timing check limits by setting them to zero. By default negative timing check limits are enabled. This is the opposite of Verilog-XL, where negative timing check limits are disabled by default, and are enabled with the +neg_tchk argument.

- **+no_notifier**
(optional) Disables the toggling of the notifier register argument of all timing check system tasks. 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.
- **+no_path_edge**
(optional) Causes ModelSim to ignore the input edge specified in a path delay. The result 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**
(optional) Disables the warning message for specify path pulse errors. 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** arguments. A path pulse error results in a warning message, and the pulse is propagated as an X. The **+no_pulse_msg** argument disables the warning message, but the X is still propagated.
- **-no_risefall_delaynets**
(optional) Disables the default rise/fall delay net delay negative timing check algorithm. This argument returns ModelSim to older behavior, where violation regions must overlap in order to find a delay net solution. Beginning with Release 6.0, this argument is unnecessary, because ModelSim uses separate rise/fall delays, so violation regions need not overlap for a delay solution to be found.
- **+no_show_cancelled_e**
(optional) Filters negative pulses on specify path delays so they do not show on the output. Default. Use **+show_cancelled_e** to drive a pulse error state.
- **+no_tchk_msg**
(optional) Disables error messages issued by timing check system tasks when timing check violations occur. Notifier registers are still toggled and may result in the propagation of Xs for timing check violations.
- **-nodpiexports**
(optional) Instructs ModelSim to not generate C wrapper code for DPI export task and function routines found at elaboration time. Does not generate the `exportwrapper.so` shared object file.
For a description of when you should use this argument, refer to “[Deprecated Legacy DPI Flows](#)” in the User’s Manual.
- **+nosdferror**
(optional) 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.

- **+nosdfwarn**
(optional) Disables warnings from the SDF annotator.
- **+nospecify**
(optional) Disables specify path delays and timing checks. Pass this argument to vopt instead of vsim when using the three-step flow.
- **+nowarnBSOB**
(optional) Disables run-time warning messages for bit-selects in initial blocks that are out of bounds.
- **+nowarn<CODE | number>**
(optional) Disables warning messages in the category specified by a warning code or number. 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>.
```

The warning code for this example is TFMPC, and the warning number is 3017. Therefore, this warning message can be disabled with +nowarnTFMPC or +nowarn3017.

- **+ntc_warn**
(optional) Enables warning messages from the negative timing constraint algorithm. 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 repeats until a solution is found. A warning message is issued for each negative limit set to zero.
- **+ntcnatchks**
(optional) Instructs vsim not to simulate timing checks, but to still consider negative timing check limits for the calculation of delayed input delays.
- **-oldvhdlforgennames**
(optional) Enables the use of a previous style of naming in VHDL for ... generate statement iteration names in the design hierarchy. The GenerateFormat value controls the previous style. The default behavior is to use the current style names. This argument duplicates the function of the OldVhdlForGenNames variable in modelsim.ini, and overrides the setting of that variable if it specifies the current style. Refer to [GenerateFormat](#), “[Naming Behavior of VHDL For Generate Blocks](#),” and [OldVhdlForGenNames](#) in the User’s Manual.
- **-onfinish ask | stop | exit | final**
(optional) Customizes the simulator shutdown behavior when it encounters \$finish in the design:
 - ask —

- In batch mode, the simulation exits.
- In GUI mode, a dialog box pops up and asks you to confirmation whether to quit the simulation.
 - stop — Stops simulation and leaves the simulation kernel running.
 - exit — Exits out of the simulation without a prompt.
 - final — Executes all final blocks then exits the simulation.

By default, the simulator exits in batch mode, and prompts you in GUI mode. Edit the **OnFinish** variable in the *modelsim.ini* file to set the default operation of \$finish. Refer to [OnFinish](#) in the User's Manual.

- **-pedanticerrors**

(optional) Forces display of an error message (rather than a warning) on a variety of conditions.

You can view a complete list of errors by executing the command:

```
verror -kind vsim -pedanticerrors
```

- **-permit_unmatched_virtual_intf**

(optional) Enables vsim to elaborate designs that have virtual interface declarations, but do not have actual interface instances that are compatible with the declarations. Such virtual interface declarations are considered "unmatched" since there is no matching or compatible interface instance. By default, unmatched virtual interfaces prevent vsim from elaborating the design. For further information on this design situation, see "Unmatched Virtual Interface Declarations" in the User's Manual.

- **-pli "<object list>"**

(optional) Loads a comma- or space-separated list of PLI shared objects. The list must be enclosed in quotes 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](#) in the User's Manual.) You can use environment variables as part of the path.

Optionally, a shared object file entry can contain an initialization function specifier:

<filename>:<init_function>

Use a colon (:) to separate the shared object filename from the initialization function name. Windows systems interpret a colon in the first 3 characters of the *<filename>* as a volume designator. In UNIX, a colon is not reserved and can be specified in the *<filename>*, but a colon is not allowed in the *<init_function>*. For this reason, the rightmost ':' is interpreted to be the separator.

To allow any number of ':' characters in the *<filename>*, a terminating ':' must be added to designate a separator to the end of the argument.

For example, a UNIX directory named "my:dir" containing a shared library named "mypli.sl" is specified as:

```
-pli my:dir/mypli.sl
```

Or the same shared library with an initialization function named "myinit" is specified in UNIX as:

```
-pli my:dir/mypli.sl:myinit
```

- **+<plusarg>**

(optional) Makes the argument following the plus sign (+) accessible to the Verilog PLI routine mc_scan_plusargs(). All plusarg argument values can be overridden in -restore mode and will take effect when simulation resumes after restoring the design.

- **+pulse_e/<percent>**

(optional) 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.

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 you do not specify the rejection limit, 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**

(optional) Selects the "on detect" style of propagating pulse errors (see +pulse_e). 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. The "on event" style is the default for propagating pulse errors (see +pulse_e_style_onevent).

- **+pulse_e_style_onevent**

(optional) 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>**

(optional) 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.

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**
(optional) 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.
- **+show_cancelled_e**
(optional) Drives a pulse error state ('X') for the duration of a negative pulse on a specify path delay. By default ModelSim filters negative pulses.
- **-showlibsearchpath**
(optional) Returns to the transcript all libraries that will be searched for precompiled modules.
- **-sv_lib <shared_obj>**
(required for use with DPI import libraries) Specifies the name of the DPI shared object with no extension. Refer to “[DPI Use Flow](#)” in the User’s Manual for additional information.
- **-sv_liblist <filename>**
(optional) Specifies the name of a bootstrap file containing the names of DPI shared objects (libraries) to load. Refer to “[DPI File Loading](#)” in the User’s Manual for format information.
- **-sv_root <dirname>**
(optional) Specifies the directory name to use as the prefix for DPI shared object lookups.
- **+transport_int_delays**
(optional) Selects transport mode with pulse control for single-source nets (one interconnect path). 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 argument works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source can be VITAL or Verilog. This argument works independently from **+multisource_int_delays**.
- **+transport_path_delays**
(optional) Selects transport mode for path delays. 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 argument affects path delays only, and not primitives. Primitives always operate in inertial delay mode.
- **-v2k_int_delays**
(optional) Makes interconnect delays visible at the load module port, per the IEEE 1364-2001 spec. By default, ModelSim annotates INTERCONNECT delays in a manner compatible with Verilog-XL. If you have `$sdf_annotation()` calls in your design that are not getting executed, add the Verilog task `$sdf_done()` after your last `$sdf_annotation()` to remove any zero-delay MIPDs that may have been created. You can use this in tandem with the **+multisource_int_delays** argument.

- **-work <pathname>**
(optional) When using a 2-step flow, overrides the library in which vsim writes the optimized design generated by the internally invoked opt command.
- **-wrealdefaultzero**
(optional) For nets declared as wreal, sets the default value for an undriven wreal net to zero (0).
- **-wreal_resolution <val>**
(optional) For Verilog nets declared as wreal, selects the resolution function that ModelSim uses to compute the effective value for multiple drivers on wreal nets. Conflicting real values on a wreal net are passed through a resolution function, which you specify with one of the following enum values:
 - DEFAULT — Single active driver only
 - 4STATE — Verilog 4-state resolution
 - SUM — Sum of all driver values
 - AVG — Average of all driver values
 - MIN — Smallest of all driver values
 - MAX — Largest of all driver valuesThe resultant value from the computation then appears on the net.

Object Arguments

The object arguments can be a [<library_name>].<design_unit>, an .mpf file, a .wlf file, or a text file. You can specify multiple design units for Verilog modules and mixed VHDL/Verilog configurations.

- <library_name>.<design_unit>
(optional) Specifies a library and associated design unit; you can make multiple library/design unit specifications. If you do not specify a library, the work library is used. You cannot use the wildcard character (*) for this argument. You can use environment variables. <design_unit> may be one of the following:

<configuration>	Specifies the VHDL configuration to simulate.
<module> ...	(optional) Specifies the name of one or more top-level Verilog modules to be simulated.
<entity> [(<architecture>)]	(optional) Specifies the name of the top-level VHDL entity to be simulated. 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

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>
(optional) Opens the specified project.
- <WLF_file_name>
(optional) Opens the specified dataset. 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. To change this behavior so that the Wave window contains all signals in the design, set the preference PrefWave(OpenLogAutoAddWave) to 1 (true).

- <text_file_name>
(optional) Opens the specified text file in a Source window.

Examples

- Invoke vsim on the entity, cpu, and assign 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 argument 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 arguments 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

- Enable the display of Start, End, and Elapsed time as well as a message count summary. Echoing of the command line is disabled

vsim -stats=time,-cmd,msg

- The first -stats argument is ignored. The none option of the second -stats argument disables all modelsim.ini settings and then enables the perf option.

vsim -stats=time,cmd,msg -stats=none,perf

vsim<info>

Returns information about the current vsim executable.

Syntax

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. You can obtain this same information with the -version argument of the **vsim** command.

Arguments

none

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.

Examples

- Interrupt a simulation, then restart it from the point of interruption.

```
vsim_break
run -continue
```

vsource

Specifies an alternative file to use for the current source file, when the current source file has been moved. The alternative source mapping applies to the current simulation only.

Syntax

`vsource [<filename>]`

Arguments

- `<filename>`

(optional) Specifies a relative or full pathname. If you omit the filename, the source file for the current design context is displayed.

Examples

```
vsource design.vhd
vsource /old/design.vhd
```

wave

A collection of related commands that manipulate and report on the Wave window.

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>]
```

Description

The following tables summarize the available options for manipulating cursors, for zooming, and for adjusting the wave display view in the Wave window:

Table 2-9. Wave Window Commands for Cursor

Curs or Com man ds	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 to show the specified or active cursor 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 2-10. 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 steps over the full range of the simulation, from time 0 to the current time: <ul style="list-style-type: none">• Delta time steps if Delta Time mode is currently selected (WLFCollapseMode = 1).• Event time steps if Event Time mode is currently selected (WLFCollapseMode = 0).
wave expand cursor	Expands simulation time into steps at the simulation time of the active cursor: <ul style="list-style-type: none">• Delta time steps if Delta Time mode is currently selected (WLFCollapseMode = 1).• Event time steps if Event Time mode is currently selected (WLFCollapseMode = 0).
wave expand range	Expands simulation time into steps over a time range specified by a start time and an end time: <ul style="list-style-type: none">• Delta time steps if Delta Time mode is currently selected (WLFCollapseMode = 1).• Event time steps if Event Time mode is currently selected (WLFCollapseMode = 0).
wave collapse all	Collapses simulation time over the full range of the simulation from time 0 to the current time.

Table 2-10. Wave Window Commands for Expanded Time Display (cont.)

Display view Commands	Description
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 2-11. 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 so 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 setitime	Positions the wave display so 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 2-12. Wave Window Commands for Zooming

Zoo ming Com man ds	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.

Arguments

- -at <percent>
(optional) Positions the display so 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 Wave window, and 100 is the right edge.
- <cursor-num>
(optional) Specifies a cursor number. If not specified, the active cursor is used.
- <factor>
(optional) Specifies how much to zoom into or out of the wave display. Default value is 2.0.
- -lock <0 |1>
(optional) Specifies the lock state of the cursor.
 0 — (default) Unlocked
 1 — Locked

- **-name <name>**
(optional) Specifies the name of the cursor.
 <name> — Any string, where the default is "Cursor <n>" and <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) Specifies a value for the designated option. Currently supported options are -name, -time, and -lock. If no option is specified, reports the current value of all options.
- **<start-time> <end-time>**
(optional) Specifies the start-time and end-time for 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) Specifies to use a Wave window other than the currently active Wave window. By default, commands run in the currently active Wave window.
 <win> — Specifies the name of a Wave window other than the currently 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

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.

Syntax

All waveforms

```
wave create [-driver {freeze | deposit | driver | expectedoutput}] [-initialvalue <value>]
[-language {vhdl | verilog}] [-portmode {input | output | inout | internal}] [-range <msb
lsb>]
[-starttime {<time><unit>}] [-endtime {<time><unit>}] <object_name>
```

Clock patterns

```
wave create -pattern clock [-dutycycle <value>] [-period {<time><unit>}] <object_name>
```

Constant patterns

```
wave create -pattern constant [-initialvalue <value>] [-value <value>] <object_name>
```

Random patterns

```
wave create -pattern random [-initialvalue <value>] [-period {<time><unit>}]
[-random_type {normal | uniform | poisson | exponential}] [-seed <value>] <object_name>
```

Repeater patterns

```
wave create -pattern repeater [-initialvalue <value>] [-period {<time><unit>}]
[-repeat {forever | never | <n>}] [-sequence {<val1>} <val2> ...]
```

Counter patterns

```
wave create -pattern counter [-direction {up | down | upthendown | downthenup}]
[-initialvalue <value>] [-period {<time><unit>}] [-repeat {forever | never | <n>}]
[-startvalue <value>] [-endvalue <value>] [-step <value>]
[-type {binary | gray | johnson | onehot | range | zerohot}] <object_name>
```

No pattern

```
wave create -pattern none <object_name>
```

Description

Refer to “Generating Stimulus with Waveform Editor” in the User’s Manual for more information.

The following table summarizes the available waveform pattern options:

Comma	Description
nd	wave create -pattern clock Generates a clock waveform. Recommended that you specify an initial value, duty cycle, and clock period for the waveform.

Comma nd	Description
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 do not specify a seed value, ModelSim 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.

Arguments

- **-pattern** clock | constant | random | repeater | counter | none
(required) Specifies the waveform pattern. Refer to “[Accessing the Create Pattern Wizard](#)” in the User’s Manual 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.
- **-dutycycle <value>**
(optional, recommended for -pattern clock) Specifies the duty cycle of the clock. Expressed as a percentage of the period during which 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 you specify <unit>, 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 is 000 and the end value is 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 to use 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 you specify <unit>, you must enclose <time> and <unit> within curly braces ({}).
- **-portmode {input | output | 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 — (default) 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. You must specify both.
- **-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. Enter multiple values as a space-separated list 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 you specify <unit>, 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 (for example, for a 3-bit signal, the start value is 000 and the end value is 111).
 <value> — Value must be appropriate for the type of waveform you are creating.
- **-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.
- **-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.

- <object_name>
(required) User specified name for the waveform. Must be the final argument.

Examples

- Create a clock signal with the following default values:
wave create -pattern clock -period 100 -dutycycle 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 \[ModelSim User's Manual\]](#)

[Accessing the Create Pattern Wizard \[ModelSim User's Manual\]](#)

wave edit

Modifies waveforms created with the wave create command.

Syntax

```
wave edit {cut | copy | paste | invert | mirror} -end {<time><unit>} -start {<time><unit>}  
    <object_name>  
wave edit insert_pulse [-duration {<time><unit>}] -start {<time><unit>} <object_name>  
wave edit delete -time {<time><unit>} <object_name>  
wave edit stretch | move {-backward {<time><unit>} | -forward {<time><unit>}}  
    -time {<time><unit>} <object_name>  
wave edit change_value -end {<time><unit>} -start {<time><unit>} <value> <object_name>  
wave edit extend -extend to | by -time {<time><unit>}  
wave edit driveType -driver freeze | deposit | driver | expectedoutput -end {<time><unit>}  
    -start {<time><unit>}  
wave edit undo <number>  
wave edit redo <number>
```

Description

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; does not affect waveform duration
wave edit delete	Delete an edge from a waveform; does not 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

Arguments

- **-backward {<time><unit>}**
(required if -forward <time> is not specified) The amount to stretch or move the edge backwards in simulation time.
 - <time> — Specified as an integer or decimal number.
 - <unit> — (optional) A suffix specifying a unit of time. 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 you specify <unit>, you must enclose <time> and <unit> within curly braces ({}).
- **cut | copy | paste | invert | mirror**
(required) Specifies the type of edit to perform.
 - cut — Deletes the specified portion of the waveform.
 - copy — Saves a copy of the specified portion of the waveform.
 - paste — Inserts the contents of the clipboard into the specified portion of the waveform.
 - invert — Flips the specified portion of the waveform vertically.
 - mirror — Flips the specified portion of the waveform horizontally.
- **-driver freeze | deposit | driver | expectedoutput**
(required) Specifies the type of driver to change the specified section of the waveform to. Applies to signals of type inout or internal.
- **-duration {<time><unit>}**
(optional) The length of the pulse.
 - <time> — Specified as an integer or decimal number. The default is 10 time units.
 - <unit> — (optional) A suffix specifying a unit of time. 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 you specify <unit>, you must enclose <time> and <unit> within curly braces ({}).
- **-end {<time><unit>}**
(required unless specifying paste) Specifies a simulation time denoting the end of the section of waveform on which to perform the editing operation.
 - <time> — Specified as an integer or decimal number.
 - <unit> — (optional) A suffix specifying a unit of time. 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 ({}).
- **-extend to | by**
(required) Specifies the format for extending waves.
 - to — Extends the wave to the time specified by -time <time>.

by — Extends the wave by the amount of time specified by -time <time>.

- -forward {<time><unit>}

(required if -backward <time> is not specified) The amount to stretch or move the edge forwards in simulation time.

<time> — Specified as an integer or decimal number.

<unit> — (optional) A suffix specifying a unit of time. 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 you specify <unit>, you must enclose <time> and <unit> within curly braces ({}).

- <number>

(optional) The number of editing operations to undo or redo. If omitted, only one editing operation is undone or redone.

- <object_name>

(required) The pathname of the waveform to edit. Must be the final argument to wave edit.

- -start {<time><unit>}

(required) Specifies a simulation time denoting the beginning of the section of waveform on which to perform the editing operation.

<time> — Specified as an integer or decimal number.

<unit> — (optional) A suffix specifying a unit of time. 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 you specify <unit>, you must enclose <time> and <unit> within curly braces ({}).

- -time {<time><unit>}

(required) The amount of time to extend or stretch waves.

<time> — Specified as an integer or decimal number.

<unit> — (optional) A suffix specifying a unit of time. 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 you specify <unit>, you must enclose <time> and <unit> within curly braces ({}).

- <value>

(required) The new value. Must match the type of the <object_name>.

Related Topics

[wave create](#)

[Generating Stimulus with Waveform Editor \[ModelSim User's Manual\]](#)

wave export

Creates a stimulus file from waveforms created with the wave create command.

Syntax

```
wave export -designunit <name> -starttime {<time><unit>} -endtime {<time><unit>} -file <filename> {-format force | vcd | vhdl | verilog}
```

Arguments

- **-designunit <name>**

(required) Specifies a design unit from which to export created waves. If you omit this argument, the command exports waves from the active design unit.

 <name> — Specifies a design unit in the simulation.

- **-endtime {<time><unit>}**

(required) The simulation time at which to stop exporting.

 <time> — Specified as an integer or decimal number.

 <unit> — (optional) A suffix specifying a unit of time. 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 you specify <unit>, you must enclose <time> and <unit> within curly braces ({}).

- **-file <filename>**

(required) The filename for the saved export file.

 <name> — Any user specified string.

- **-format force | vcd | vhdl | verilog**

(required) The format of the saved stimulus file. The format options include:

 force — A Tcl script that recreates the waveforms. The file must 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><unit>}**

(required) The simulation time at which to start exporting.

 <time> — Specified as an integer or decimal number.

 <unit> — (optional) A suffix specifying a unit of time. 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 you specify <unit>, you must enclose <time> and <unit> within curly braces ({}).

Related Topics

[wave create](#)

[wave import](#)

[Generating Stimulus with Waveform Editor \[ModelSim User's Manual\]](#)

wave import

Imports an extended VCD file that was created with the wave export command. It cannot read extended VCD files 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>`
(required) The name of the extended VCD file to import.

Related Topics

[wave create](#)

[wave export](#)

[Generating Stimulus with Waveform Editor \[ModelSim User's Manual\]](#)

wave modify

Modifies waveform parameters set by a previous wave create command.

Syntax

All waveforms

```
wave modify [-driver freeze | deposit | driver | expectedoutput] [-endtime {<time><unit>}]  
[-initialvalue <value>] [-portmode {input | output | inout | internal}] [-range <msb lsb>]  
[-starttime {<time><unit>}] <wave_name>
```

Clock patterns only

```
wave modify -pattern clock -period <value> -dutycycle <value> <wave_name>
```

Constant patterns only

```
wave modify -pattern constant [-driver freeze | deposit | driver | expectedoutput]  
[-language {vhdl | verilog}] [-value <value>] <wave_name>
```

Counter patterns only

```
wave modify -pattern counter -period <value> -repeat forever | <n> | never -startvalue <value> -  
step <value> [-direction {up | down | upthendown | downthenup}]  
[-endvalue <value>] [-type {binary | gray | johnson | onehot | range | zerohot}]  
<wave_name>
```

Random patterns only

```
wave modify -pattern random -period <value>  
-random_type exponential | normal | poisson | uniform [-seed <value>] <wave_name>
```

Repeater patterns only

```
wave modify -pattern repeater -period <value> -repeat forever | <n> | never  
-sequence {val1 val2 val3 ...} <wave_name>
```

No pattern

```
wave create -pattern none <wave_name>
```

Description

The following table summarizes the available wave modification options:

Comma nd	Description
wave modify - pattern clock	Generates a clock waveform. Specify an initial value, duty cycle, and clock period for the waveform.

Comma nd	Description
wave modify - pattern constant	Generates a waveform with a constant value. Specify a value.
wave modify - 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 modify - 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 do not specify a seed value, ModelSim uses a default value of 5.
wave modify - 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 modify - pattern none	Creates a placeholder for a waveform. Specify an object name.

Arguments

- **-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 the type of the signal driver. Applies to signals of type inout or internal.

- **-dutycycle <value>**
(required) 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) Specifies the simulation time when the waveform should 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. 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 you specify <unit>, you must enclose <time> and <unit> within curly braces ({}).
- **-endvalue <value>**
(optional) Specifies 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. For example, for a 3-bit signal, the start value is 000 and end value is 111.
 <value> — Any positive integer.
- **-initialvalue <value>**
(optional) The initial value for the waveform. Value must be appropriate for the type of waveform you are creating. Not applicable to counter patterns.
 <value> — Any positive integer.
- **-language {vhdl | verilog}**
(optional) Controls which language to use to modify the wave.
 vhdl — (default) Specifies the VHDL language.
 verilog — Specifies the Verilog language.
- **-period <value>**
(required) The period of the signal.
- **-portmode {input | output | inout | internal}**
(optional) The port type for the waveform.
 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 exponential | normal | poisson | uniform**
(required) Specifies a random pattern to generate.

exponential — Exponential distribution of waveform events.

normal — Normal or Gaussian distribution of waveform events.

poisson — Poisson distribution of waveform events.

uniform — (default) Uniform distribution of waveform events.

- **-range <msb lsb>**

(optional) Identifies bit significance in a counter pattern.

msb lsb — Most significant bit and least significant bit. You must specify both.

- **-repeat forever | <n> | never**

(required) Controls duration of pattern repetition.

forever — Repeat the pattern for as long as the simulation runs.

<n> — Repeat the pattern <n> number of times, where <n> is any positive integer.

never — Never repeat the pattern during simulation.

- **-seed <value>**

(optional) Specifies a seed value for a randomly generated waveform.

<value> — Any non-negative integer. The default is 5.

- **-sequence {val1 val2 val3 ...}**

(required) The set of values that you want repeated.

<val1> — Value must be appropriate for the type of waveform you are creating. Enter multiple values as a space-separated list, enclosed in curly braces ({}).

- **-starttime {<time><unit>}**

(optional) The simulation time at which to start the waveform. 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. 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 you specify <unit>, 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. For example, for a 3-bit signal, the start value is 000 and the end value is 111.

<value> — Value must be appropriate for the type of waveform you are creating.

- **-step <value>**

(required) The step by which the counter is incremented/decremented.

<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.
- <wave_name>
(required) The name of an existing waveform created with the [wave create](#) command.

Related Topics

[wave create](#)

[Generating Stimulus with Waveform Editor \[ModelSim User's Manual\]](#)

[Accessing the Create Pattern Wizard \[ModelSim User's Manual\]](#)

wave sort

Sorts signals in the Wave window by name or full path name.

Syntax

wave sort {ascending | descending | fa | fd}

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.

fd — Sort in descending order by full path name.

Examples

wave sort ascending

when

Instructs ModelSim to perform actions when specified conditions are met.

Syntax

```
when [[-fast] [-id <id#>] [-label <label>] [-repeat] {<when_condition_expression>} {<command>}]
```

Description

Use this command to control ModelSim activity for one or more specified conditions.

For example, you can use the command to break on a signal value, or at a specific simulator time. Use the [nowhen](#) command to deactivate when commands.

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, not related to Tcl, which permits only the operators listed in [Table 2-13](#), and operands that can be HDL object names, signame'event, or constants. ModelSim evaluates the condition every time any object in the condition changes, hence the restrictions.

The following items can affect your use of 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.
- You cannot use virtual signals, functions, regions, types, and so forth, in the when command. You also cannot use simulator state variables other than \$now.
- You can enter when with no arguments to list the currently active when statements and their labels (explicit or implicit).

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 must use the Tcl uivar command to declare the state. 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

You cannot access channels (such as files, pipes, sockets) that were opened outside of the embedded command. 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 use of 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 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
}
```

Generally, any embedded command done using the -fast argument is global to all other commands used with the -fast argument. In the example above, {\$now == 0ns} defines Tcl processes in a way that the -fast commands can use. These processes have the same restrictions that when bodies have, but the advantage is speed, as a proc tends to execute faster than code in the when body.

It is recommended not to use virtual signals and expressions.

Arguments

- **-fast**
(optional) Causes the embedded <command> to execute in the simulation kernel, which provides faster execution and reduces impact on simulation runtime performance.
Limitations on using the -fast argument are described above (in “[Embedded Commands Not Allowed with the -fast Argument](#)”). The disallowed commands still work, but they slow down the simulation.
- **-label <label>**
(optional) Used to identify individual when commands.

<label> — Associates a name or label with the specified when command, adding a level of identification. The label can contain special characters. You must use quotation marks (" ") or braces ({ }) to enclose any <label> that contains spaces or special characters.
- **-id <id#>**
(optional) Attempts to assign this id number to the when command.

<id#> — Any positive integer that is not already assigned. If the id number you specify is already in use, ModelSim returns 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 in use for a breakpoint.

- -repeat

(Limited to “when” breakpoint expressions involving “\$now”). Instructs the command to reestablish the breakpoint after it has been triggered, so that it will fire again for the next time period. Without this argument, expressions using \$now trigger only once.

- {<when_condition_expression>}

(required if a command is specified) Specifies the conditions to be met in order to execute the specified <command>. The condition is evaluated in the simulator kernel and can be an object name, in which case you can omit the curly braces. The command executes when the object changes value. The condition can be an expression with these operators:

Table 2-13. when_condition_expression Operators

Name	Operator
equals	==, =
not equal	!=, /=
greater than	>
less than	<
greater than or equal	>=
less than or equal	<=
AND	&&, AND
OR	, OR

The operands can be object names, signame'event, or constants. Subexpressions in parentheses are permitted. The command executes 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; you cannot compare the value of two signals. For example, `Name = Name` is not possible.

You can use Tcl variables 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, so 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 " $>$ ", " $<$ ", " \geq ", and " \leq " operators are the standard for vector types, not the overloaded operators in the `std_logic_1164` package. This can 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>}

(required if a when expression is specified) The Tcl interpreter within the ModelSim GUI evaluates command(s) for this argument. Any ModelSim or Tcl command or series of commands are valid with one exception—you cannot use the `run` command with the when command. The command sequence usually contains a `stop` command that sets a flag to break the simulation run after completion of the command sequence. You can use multiple-line commands.

Note

 If you want to stop the simulation with 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 when command below echoes the simulator time when slice [3:1] of wire [15:0] count matches the hexadecimal value 7, and simulation time is between 70 and 111 nanoseconds.

```
when {$now > 70ns and count(3:1) == 3'h7 && $now < 111ns} {  
    echo "count(3:1) matched 3'h7 at time " $now  
}
```

- 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 clk_b_path /tb/ps/dprb_0/udprb/ucar_reg/uint_ram/clk_b;  
when -label when1 "$clk_b_path'event and $clk_b_path ='1" {  
    echo "Detected Clk edge at path $clk_b_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}
```

- 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 so that the time unit can 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 to:

```
when "0 = 1000" stop
```

where

Displays information about the system environment. Useful for debugging when ModelSim cannot find the required libraries or support files.

Syntax

where

Description

The command displays two results on consecutive lines:

- current directory

This is the current directory from which ModelSim was invoked, 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. For designs that were not loaded through a project, this line displays the *modelsim.ini* file in the current directory.

Arguments

None.

Examples

- Design is loaded through a project:

VSIM> where

Returns:

```
# 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

Returns:

```
# Current directory is: C:\Client\testcase\mydesign
# Project is: modelsim.ini
```

wlf2log

Translates a ModelSim WLF file (*vsim.wlf*) to a QuickSim II logfile. It 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



Invoke this command only after you have stopped the simulation using [quit -sim](#) or [dataset close sim](#).

Syntax

```
wlf2log <wlffile> [-bits] [-fullname] [-help] [-inout] [-input] [-internal] [-l <instance_path>]  
[-lower] [-o <outfile>] [-output] [-quiet]
```

Arguments

- <wlffile>
(required) Specifies the ModelSim WLF file to convert.
- -bits
(optional) Forces vector nets to be split into 1-bit wide nets in the log file.
- -fullname
(optional) Shows the full hierarchical pathname when displaying signal names.
- -help
(optional) Displays a list of command options, with a brief description for each.
- -inout
(optional) Lists only the inout ports. You can combine this argument with the -input, -output, or -internal arguments.
- -input
(optional) Lists only the input ports. You can combine this argument with the -output, -inout, or -internal arguments.
- -internal
(optional) Lists only the internal signals. You can combine this argument with the -input, -output, or -inout arguments.
- -l <instance_path>
(optional) Lists the signals at or below an HDL instance path within the design hierarchy.
 <instance_path> — Specifies an HDL instance path.

- **-lower**
(optional) Shows all logged signals in the hierarchy. When invoked without the -lower switch, displays only the top-level signals.
- **-o <outfile>**
(optional) Directs the output to be written to a file, where the default destination for the logfile is standard out.
 <outfile> — A user specified filename.
- **-output**
(optional) Lists only the output ports. You can combine this argument with the -input, -inout, or -internal arguments.
- **-quiet**
(optional) Disables error message reporting.

wlf2vcf

Translates a ModelSim WLF file to a standard VCD file. Complex data types that are unsupported in the VCD standard (records, memories, and so on) are not converted.

Note

 Invoke this command only after you have stopped the simulation using [quit -sim](#) or [dataset close sim](#).

Syntax

```
wlf2vcf <wlffile> [-help] [-o <outfile>] [-quiet]
```

Arguments

- <wlffile>
(required) Specifies the ModelSim WLF file to convert.
- -help
(optional) Displays a list of command options, with a brief description for each.
- -o <outfile>
(optional) Specifies a filename for the output, where the default destination for the VCD output is stdout.
 <outfile> — A user specified filename.
- -quiet
(optional) Disables warning messages that are produced when an unsupported type (for example, records) is encountered in the WLF file.

wlfman

A collection of related commands that enable you to get information about, and perform various actions on, saved WLF files.

Syntax

```
wlfman info <source_wlffile> [-v]
wlfman items <source_wlffile> [-n] [-v]
wlfman filter -o <out_wlffile> <source_wlffile> [-begin <time>] [-end <time>]
    [-collapsedelta | -collapsetime | -nocollapse] [-compress | -nocompress]
    [-f <object_list_file>] [-index | -noindex] [-r <object>] [-nowarn <number>] [-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 monitor [-f | -i <intervalTime> | -p <endTime>] [-q | -v] <source_wlffile>
wlfman optimize -o <out_wlffile> <source_wlffile> [-compress | -nocompress]
    [-index | -noindex] [-opt | -noopt]
```

Description

The different wlfman commands perform the following functions on saved WLF files:

- wlfman info — Returns file information, resolution, versions, and so forth about the source WLF file.
- wlfman items — Generates a list of HDL objects (that is, signals) and/or transaction streams from the source WLF file, and outputs it to stdout. When redirected to a file, the output is referred to as an object_list_file, and can be read in by wlfman filter. Following is an example of an object_list_file:

```
/top/foo # signal foo
/top/u1/* # all signals under u1
/top/u1 # same as line above
-r /top/u2 # recursively, all signals under u2
/top/stream1 # transaction stream stream1
```

The object_list_file is a list of objects and/or transaction streams, 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. Transaction object lists can include a stream name, stream array name, or sub stream, all with a full path. Transaction objects recorded in the object_list_file include:

- Full path of a stream array — Logs and records all of the individual streams in the specified stream array, the sub-streams of each stream, and the phase sub-streams

recursively for each sub-stream, including the attributes of the transactions in the sub-streams and phase sub-streams. For example:

```
/top/<stream_array> # stream array full path (ex: /top/stream_array)
```

- Full path of a stream object — Logs and records all of the sub-streams and recursively records all phase sub-streams for each specified sub-stream including the attributes of the transactions present in the sub-streams and phase sub-streams. For example:

```
/top/<stream_object> # individual stream full path (ex: /top/stream)
```

- Sub-streams — Logs and records all of the attributes of the specified sub-stream. Other sub-streams of the main stream are not logged. Phase sub-streams cannot be individually logged. For example:

```
/top/<stream_object> # individual stream full path (ex: /top/stream)
```

```
/top/<stream_object>.<sub-stream> # sub-stream full path (ex: /top/stream.s0)
```

Note



You can produce these files from scratch you must use the correct syntax. It is recommended that you use wlfman items as it 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 monitor — Returns the current state of a WLF file to the transcript. Outputs a new line of information each time the state is monitored. The state of the WLF file can be monitored at regular intervals, indicating the changes over time. For example:

```
wlfman monitor visim.wlf
File      Sim
State     Time
closed   14000
```

- wlfman profile — Generates a report with an estimate of the percentage of file space 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 (such as a logged memory that uses a zero-delay integer loop to initialize the memory). You may be able to reduce WLF file size by not logging those signals.

When the WLF file contains transaction streams and/or stream arrays, wlfman profile generates an additional report estimating the file space each transaction uses as a percentage of total file size. You may be able to reduce WLF file size by not logging some transactions or streams.

The estimated size of a transaction is equal to the size of the transaction without any user attributes, plus the sum of the sizes of every attribute in that transaction. Also, the

estimate assumes that every transaction has its attributes recorded if one of the transactions in a sub-stream has that attribute. If the object is a stream array, the sum of the sizes of all the streams is presented, not the sizes of the individual elements.

- **wlfman merge** — Combines two WLF files with different signals or transaction objects into one WLF file. It *does not* combine WLF files containing the same signals at different runtime ranges (for example, mixedhdl_0ns_100ns.wlf & mixedhdl_100ns_200ns.wlf). A merge of two WLF files that contain the same transaction streams records the first stream's data, ignores the second stream, and issues a warning that a horizontal merge is not supported.
- **wlfman optimize** — Copies the data from the WLF file to the output WLF file, adding or replacing the indexing and optimization information.

The wlfman commands are designed to be used in combination. For example, if you run wlfman profile, and identify that an unneeded signal or transaction stream accounts for 50% of the WLF file size, you can then run wlfman filter to remove the object from the WLF file.

Arguments

- **-o <out_wlffile>**
(required) Specifies the name of the output WLF file. The output WLF file contains 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>. Transactions that start prior to the specified time are ignored.
- **-collapsedelta | -collapsetime | -nocollapse**
(optional) Controls preservation of events in the resulting WLF file. The data preserved depends on how events were recorded in the input WLF file. Specifying a finer granularity of preservation than the input WLF file has no additional affect.
 - collapsedelta — (default) Preserves only the values at the end of a delta.
 - collapsetime — Preserves only the values at the end of a time step.
 - nocollapse — Preserves all events.
- **-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 filtering of <source_wlffile> stops and no further data is logged.
- -f
(optional) Repeat status update every 10 seconds of real time, unless an alternate time interval is specified with -i <intervalTime>.
- -f <object_list_file>
(optional) Specifies an object_list_file, created by wlfman items or by the user, to include in <out_wlffile>. For user created object list files, the object list can include stream name, stream array name, or sub stream with a full path.
- -i <intervalTime>
(optional) Specifies the time delay before the next status update. The default is 10 seconds of real time, if not specified.
 <intervalTime> — Any positive integer.
- -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. You can add indexing and optimization information back to the file with the wlfman optimize command.
 - index — Enables indexing. (default)
 - noindex — Disables indexing and optimization.
- -n
(optional) Lists regions only (no signals).
- -nowarn <number>
(optional) Selectively disables a category of warning messages.
 - 1 — Disables "Skipping unsupported object" warning message.
- -opt | -noopt
(optional) Controls optimization of the output WLF file.
 - opt — Enables WLF file optimization. (default)
 - noopt — Disables WLF file optimization.

- **-p <endTime>**
(optional) Specifies the simulation time at which wlfman will stop monitoring the WLF file.
 <endTime> — Any positive integer.
- **-q**
(optional) Suppress normal status messages while monitoring.
- **-r <object>**
(optional) Specifies an object (region) to recursively include in the output. If <object> is a signal, the output is 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. The default is to include all objects.
- **-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 to display only the top <number> signals in terms of file space percentage.
- **-v**
(optional) Produces verbose output listing the object type next to each object.

Examples

- Specifying the command:

```
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           1224     18 %  /top_vh/ptrans
#          1           1216     18 %  /top_vh/sdata
#          3           675      10 %  /top_vh/strans
#          5           423      6  %   /top_vh/cache/gen_3/s/data_out
#          6           135      3  %   /top_vh/paddr.
#
#          .
#          .
#          .
```

- Specifying the command:

wlfman profile -top 3 trans.wlf

returns:

```
#The following table lists the number of transitions and approximate
#wlf file space consumed (prior to compression) for each signal
#logged in the wlf file.
#Repeated ID #'s mean those signals share the same
#space in the wlf file.
#
# ID      Transitions   File %    Name
#-----  -----
#          1           1001     11 %  /top/t3
#          2           1001     11 %  /top/t1
#          3           1         0  %   /top/s2
#The following table lists the number of transactions and
#approximate wlf file space consumed (prior to compression) for each
#stream or stream array logged in the wlf file.
#
# ID      Transactions   File %    Name
#-----  -----
#          1           3000     61 %  /top/stream2
#          1           1000     17 %  /top/stream1
```

- Specifying the command:

wlfman monitor -f -p 1000000000 vsim.wlf

Returns:

```
Setting end time to 100000000, measuring progress %
File      File      Percent
State     Time      Complete
open      7239185   7.2%
open      7691785   7.7%
open      8144385   8.1%
open      8596625   8.6%
```

Related Topics

[Recording Simulation Results With Datasets \[ModelSim User's Manual\]](#)

[WLF File Parameter Overview \[ModelSim User's Manual\]](#)

wlfrecover

Attempts to "repair" WLF files that are incomplete because of a crash, or because the file was copied prior to completion of the simulation. Use this command if you receive a "bad magic number" error message when opening a WLF file. You can run the command from the VSIM> or ModelSim> prompt, or from a shell.

Syntax

wlfrecover <filename> [-force] [-q]

Arguments

- <filename>
(required) Specifies the WLF file to repair.
- -force
(optional) Disregards file locking, and attempts to repair the file.
- -q
(optional) Hides all messages, unless there is an error while repairing the file.

Related Topics

[Saving a Simulation to a WLF File \[ModelSim User's Manual\]](#)

write format

This 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.

Syntax

```
write format {<window_type>} <filename>  
write format restart [<option option1 ...>] <filename>
```

Description

The write command creates a file that is primarily a list of [add list](#) or [add wave](#) commands, but it includes a few other commands (refer to "Output" below).

You can invoke this command with the [do](#) command to recreate the window format on a subsequent simulation run (refer to [restart](#) below).

Arguments

- <window_type>
(required unless specifying restart) Specifies to record the contents of <window_type> in the file designated by <filename>.
 - breakpoints — Records file line and signal breakpoints.
 - list — Records objects of the List window.
 - memory — Records objects of the Memory window.
 - msgviewer — Records objects of the Message Viewer window.
 - watch — Records objects of the Watch window.
 - wave — Records objects of the Wave window.
 - restart — Records objects of all windows and breakpoints in the .do file.
- restart
(required) Creates a .do file that records all debug windows, all file/line breakpoints, and all signal breakpoints created using the [when](#) command. The ShutdownFile *modelsim.ini* variable, when set to this .do filename, calls the write format restart command upon exit. Refer to [ShutdownFile](#) in the User's Manual.

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 when not all datasets exist, use the -force switch with your do command. For example:

```
VSIM> do format.do -force
```

Note

 Note that if you use the -force switch when some datasets do not exist, you will get error messages for signals referencing the nonexistent datasets. Also, -force is recognized by the format file, not the do command.

- <option option1 ...>
(optional) Excludes a specific type of information from write format restart .do file.
 - -nobreak — Do not record breakpoints.
 - -nolastnow — Do not report last now value.
 - -nolist — Do not record the List window format.
 - -nomemory — Do not record Memory window views.
 - -nosource — Do not record source files.
 - -novsim — Do not record the vsim command.
 - -nowave — Do not record the Wave window format.
- <filename>
(required) Specifies the name of the output file to write the data to. You must specify the .do extension.

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
```

- 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/clk
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, the -noupdate argument adds five signals 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. Only saved Wave format files use these four commands; they are not documented elsewhere.

write list

Records the contents of the List window in a list output file.

Syntax

`write list [-events] <filename>`

Description

The list output file contains simulation data for all HDL objects displayed in the List window: VHDL signals and variables, and Verilog nets and registers.

Arguments

- `-events`
(optional) Specifies to write the output in print-on-change format. The default is tabular format.
- `<filename>`
(required) Specifies the name of the output file to write the data to.

Examples

- Save the current data in the List window in a file named *alu.lst*.

`write list alu.lst`

Related Topics

[write tssi](#)

write preferences

Saves the current GUI preference settings to a Tcl preference file. The saved setting 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>

(required) Specifies the name for the preference file. 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. Refer to [MODELSIM_TCL](#) in the User's Manual.

You can modify variables by editing the preference file with the ModelSim [notepad](#):

```
notepad <preference file name>
```

write report

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.

Syntax

```
write report [-capacity [-l | -s] [-line] [-qdas | -vmem]] |  
[-l | -s] | [-tcl] | [<filename>]
```

Description

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

Arguments

- **-capacity**

(optional) Reports data on memory usage of various types of SystemVerilog constructs in the design.

ModelSim collects memory usage data for dynamic objects.

Must be specified first when specifying **-qdas**.

To display memory data for all object types, specify **-capacity -l**.

- <filename>
(optional) Specifies the name of the output file to write the data to. If you omit <filename>, the report is written to the Transcript window.
- -l
(optional) Generates detailed information about the design, including a list of sparse memories or the memory capacity for all object types. You must precede this argument with -capacity when specifying a capacity report.
- -line
(optional) Generates point of allocation (line) based report. If you do not specify -line, the report is generated based on declaration. Vsim must be run with -capacity=line to print a point-of-allocation (line) based report.
- -qdas
(optional) Reports memory usage data for queues, dynamic arrays, associative arrays, and strings (each in its own section in the report). You must precede this argument with -capacity when specifying a capacity report.
- -s
(optional) Generates a short list of design information. You must precede this argument with -capacity when specifying a capacity report.
- -tcl
(optional) Generates a Tcl list of design unit information. You cannot use this argument when generating an output file with the <filename> argument.
- -vmem
(optional) When specified with capacity, -vmem reports usage data for Verilog memories.

Examples

- Save information about the current design in a file named *alu_rpt.txt*.

write report alu_rpt.txt

- Create a Simulation Report for the current simulation

write report -l

returns:

```
##  
## 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  
...  
.
```

- Create a Tcl-based report for the current simulation:

write report -tcl

which creates output similar to:

```
{1 {Package Body} <path>/ieee_std_logic_textio {} {<path>/vhdl_src/  
synopsys/std_logic_textio.vhd} notAcell}  
{4 Entity <path>/dataflow/work cache_set only {set.vhd util.vhd}  
notAcell}  
{4 Module <path>/dataflow_verilog/work cache_set fast {set.v}  
notAcell}
```

where the output is of the format:

```
{<occurrences> <type> <library_location> <name> <architecture>  
<source_filename> ...} <cell_info>}
```

write timing

Displays path delays and timing check limits, unadjusted for delay net delays, for the specified instance.

Syntax

```
write timing [-recursive] [-file <filename>] [<instance_name1>...<instance_nameN>]  
[-simvalues]
```

Description

The write timing command reports interconnect delays on a Verilog module instance as either MIPDs (Module Input Port Delays) or MITDs (Module Transport Port Delays). If you specify either the `+multisource_int_delays` or the `+transport_int_delays` argument with the `vsim` command, INTERCONNECT delays are reported as MITDs. Otherwise they are reported as MIPDs.

Following is an example MIPD report:

```
# /top/u1: [mymod:src/5/test.v(18)]# MIPD(s):  
#   Port clk_in: (6, 6, 6)
```

Following is an example MITD report:

```
# /top/u1: [mymod:src/5/test.v(18)]  
# MITDs to port clk_in:  
# From port /top/p/y = (6)
```

When you specify the `+multisource_int_delays` argument without `+sdf_verbose` on the `vsim` command line, "write timing" does not report the individual bits of vector source ports of SDF INTERCONNECT delays.

For example, assume the SDF file contains the following two INTERCONNECT statements:

```
(INTERCONNECT p/y[0] n/bus_in[0] (3))  
(INTERCONNECT p/y[1] n/bus_in[1] (4))
```

The corresponding "write timing" output is:

```
# MITDs to port bus_in[0]:  
# From port /tb12/p/y = (3)
```

```
# MITDs to port bus_in[1]:  
#   From port /tb12/p/y = (4)
```

Notice that the report does not include the specific bits of the source port.

If you add "+sdf_verbose" to the vsim command line, the "write timing" output becomes:

```
# MITDs to port bus_in[0]:  
# From port /tb12/p/y[0] = (3)  
# MITDs to port bus_in[1]:  
#   From port /tb12/p/y[1] = (4)
```

The report now includes the specific bits of the source port.

Arguments

- **-file <filename>**
(optional) Specifies the name of the output file where the data is to be written. If the -file argument is omitted, timing information is written to the Transcript window.
 <filename> — Any valid filename. May include special characters and numbers.
- **<instance_name1>...<instance_nameN>**
(required) The name(s) of the instance(s) for which timing information will be written. If <instance_name> is omitted, the command returns nothing.
- **-recursive**
(optional) Generates timing information for the specified instance and all instances underneath it in the design hierarchy.
- **-simvalues**
(optional) Displays optimization-adjusted values for delay net delays.

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

Writes the contents of the Transcript window to a file. You can then modify the resulting file to replay the transcribed commands as a DO file (macro).

Note

 You cannot use this command in batch mode. In batch mode, use the standard Transcript file or redirect stdout.

Syntax

`write transcript [<filename>]`

Arguments

- `<filename>`
(optional) Specifies the name of the output file to write the data to. If you omit `<filename>`, the transcript is written to a file named *transcript*.

Related Topics

[Saving a Transcript File as a DO file \[ModelSim GUI Reference Manual\]](#)

write tssi

Records the contents of the List window in a “TSSI format” file.

Note

 TSSI format is sometimes referred to as SEF format. The two terms are interchangeable.

Syntax

`write tssi <filename>`

Description

The write tssi command creates a file that contains TSSI simulation data for all HDL objects displayed in the List window that can be converted to TSSI format (VHDL signals and Verilog nets). It also generates a signal definition file.

The List window must be using symbolic radix in order for write tssi to produce useful output.

If <filename> has a file extension (for example, *listfile.lst*), the definition file uses the same filename with the extension *.def* (for example, *listfile.def*). The values appear in the listfile in the same order in which they appear in the List window. If the object is a port, the port type determines the directionality. Otherwise, the object is assumed to be bidirectional (mode INOUT).

Objects that the write tssi command can convert to TSSI format are VHDL enumerations with 255 or fewer elements and Verilog nets. The command converts enumeration values U, X, 0, 1, Z, W, L, H, and - (the enumeration values defined in the IEEE Standard 1164 std_ulogic enumeration) to TSSI values as shown in the table below.

std_ulogic State Characters	TSSI (SEF) State Characters		
	I	Output	Bidirectional
n	t	p	
U	N	X	?
X	N	X	?
0	D	L	0
1	U	H	1
Z	Z	T	F
W	N	X	?
L	D	L	0

std_ulogic State Characters	TSSI (SEF) State Characters		
	I	Output	Bidirectional
n		H	
p		U	
u		H	
t		1	
-		N	X
			?

Lowercase values x, z, w, l, and h are converted to the same values as the corresponding capitalized values. 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, it also converts any signal that uses only the values defined for std_ulogic (including the VHDL standard type bit).

Bidirectional logic values are not converted because only the resolved value is available. You can use the TSSI TDS ASCII In Converter and ASCII Out Converter to resolve the directionality of the signal, and to determine the proper forcing or expected value on the port.

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.

Arguments

- <filename>
(required) Specifies the name of the output file where the data is to be written.

write wave

Records the contents of the Wave window in PostScript format. The output file can then be printed on a PostScript printer.

Syntax

```
write wave <filename> [-end <time>] [-landscape] [-height <real_num>] [-margin <real_num>]  
[-perpage <time>] [-portrait][-start <time>] [-width <real_num>]
```

Arguments

- <filename>
(required) Specifies the name of the PostScript (.ps) output file.
- -end <time>
(optional) The simulation time at which to end the record.
 <time> — Specified as a positive integer or decimal number. The units are the current simulation time resolution.
- -height <real_num>
(optional) Specifies the paper height in inches.
 <real_num> — Specified as a positive integer or decimal number. The default is 11.0.
- -landscape
(optional) Use landscape (horizontal) orientation. (default)
- -margin <real_num>
(optional) Specifies the margin in inches.
 <real_num> — Specified as a positive integer or decimal number. The default is 0.5.
- -perpage <time>
(optional) Specifies the time width per page of output.
 <time> — Specified as a positive integer or decimal number. The units are the current simulation time resolution.
- -portrait
(optional) Use portrait (vertical) orientation. The default is landscape (horizontal).
- -start <time>
(optional) Specifies the start time to be written.
 <time> — Specified as a positive integer or decimal number. The units are the current simulation time resolution.
- -width <real_num>
(optional) Specifies the paper width in inches.

<real_num> — Specified as a positive integer or decimal number. The default is 8.5.

Examples

- Save the current data in the Wave window in a file named *alu.ps*.

write wave alu.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 ease the job of creating a PostScript waveform output file, you can use the **File > Print Postscript** menu selection in the Wave window.

Index

— Symbols —

'delayed, 36
'hasX, 36
'hasX, hasX, 36
+define+, 399
+delay_mode_distributed, 399
+delay_mode_path, 399
+delay_mode_unit, 399
+delay_mode_zero, 400
+incdir+, 403
+maxodelays, 405
+mindelays, 405
+nowarn, 409
+typdelays, 419

— A —

abort command, 65
absolute time, using @, 27
add dataflow command, 66
add list command, 68
add log command, 198
add memory command, 72
add message command, 74
add watch command, 76
add wave command, 77
add_cmdhelp command, 84
addTime command, 287
alias command, 86
analog
 signal formatting, 79
annotating interconnect delays,
 v2k_int_delays, 464
archive load command, 87
archive write command, 88
arrays
 slices, 16
arrays
 indexes, 16
assertions

testing for with onbreak command, 227
attributes, of signals, using in expressions, 34
automati c saving of coverage, 429

— B —

batch_mode command, 89
batch-mode simulations
 halting, 502
bd (breakpoint delete) command, 90
binary radix, mapping to std_logic values, 43
bookmark add wave command, 91
bookmark delete wave command, 93
bookmark goto wave command, 94
bookmark list wave command, 95
bp (breakpoint) command, 96
break
 on signal value, 495
breakpoints
 conditional, 495
 continuing simulation after, 259
 deleting, 90
 listing, 96
 setting, 96
 signal breakpoints (when statements), 495
 time-based
 in when statements, 502
busses
 user-defined, 82

— C —

case choice, must be locally static, 339
case sensitivity
 VHDL vs. Verilog, 20
cd (change directory) command, 106
change command, 107
class instance garbage collector, 191, 193
class member selection, syntax, 16
class objects, viewing, 112, 114, 116, 119, 122,
 124, 126, 128

classinfo command, 112, 114, 116, 119, 122, 124, 126, 128
Color
 radix, 245
 example, 246
combining signals, busses, 82
commands
 abort, 65
 add dataflow, 66
 add list, 68
 add memory, 72
 add message, 74
 add wave, 77
 add_cmdhelp, 84
 alias, 86
 archive load, 87
 archive write, 88
 batch_mode, 89
 bd (breakpoint delete), 90
 bookmark add wave, 91
 bookmark delete wave, 93
 bookmark goto wave, 94
 bookmark list wave, 95
 bp (breakpoint), 96
 cd (change directory), 106
 change, 107
 classinfo, 112, 114, 116, 119, 122, 124, 126, 128
 configure, 130
 dataset close, 139
 dataset current, 142
 dataset info, 143
 dataset list, 144
 dataset open, 145
 dataset rename, 146
 dataset restart, 147
 dataset save, 148
 dataset snapshot, 149
 delete, 152
 describe, 153
 disablebp, 154
 do, 155
 drivers, 157
 dumplog64, 159
 echo, 160
 edit, 161
 enablebp, 162
 encoding, 163
 environment, 164
 examine, 166
 exit, 173
 find, 175
 find connections, 180
 find infiles, 181
 find insource, 182
 force, 184
 gc configure, 191
 gc run, 193
 help, 194
 history, 195
 layout, 196
 log, 198
 lshift, 201
 lsublist, 202
 mem compare, 203
 mem display, 204
 mem list, 207
 mem load, 208
 mem save, 212
 mem search, 215
 modelsim, 218
 nolog, 220
 notepad, 222
 noview, 223
 nowhen, 224
 onbreak, 225
 onElabError, 227
 onerror, 228
 onfinish, 230
 pause, 231
 printenv, 232, 233
 process report, 234
 pwd, 238
 quietly, 239
 quit, 240
 radix, 241
 radix define, 244
 radix list, 249
 radix name, 250
 readers, 252

report, 253
restart, 255
resume, 257
run, 258
runStatus, 261
searchlog, 263
see, 266
setenv, 267
shift, 268
show, 269
simstats, 270
simstatslist, 272
stack down, 274
stack frame, 275
stack level, 276
stack up, 278
status, 279
stop, 282
suppress, 283
Time, 287
transcript, 290
transcript file, 291
transcript path, 293
transcript sizelimit, 294
TreeUpdate, 518
tssi2mti, 298
unsetenv, 301
variables referenced in, 26
vcd add, 302
vcd checkpoint, 305
vcd comment, 306
vcd dumpports, 307
vcd dumpportsall, 310
vcd dumpportsflush, 311
vcd dumpportslimit, 312
vcd dumpportsoff, 314
vcd dumpportson, 315
vcd file, 316
vcd files, 318
vcd flush, 321
vcd limit, 323
vcd off, 325
vcd on, 326
vcom, 331
vdel, 348
vdir, 350
vencrypt, 353
verror, 359
vgencomp, 361
vhencrypt, 363
virtual count, 370
virtual define, 371
virtual delete, 372
virtual describe, 373
virtual expand, 374
virtual function, 375
virtual hide, 378
virtual log, 379
virtual nohide, 381
virtual nolog, 382
virtual region, 384
virtual save, 385
virtual show, 386
virtual signal, 387
vlib, 393
vlog, 397
vmake, 423
vmap, 425
vsimVersion, 468
vsource, 470
wave, 472
wave create, 477
wave edit, 483
wave export, 486
wave import, 488
wave modify, 489
wave sort, 494
WaveActivateNextPane, 518
WaveRestoreCursors, 518
WaveRestoreZoom, 518
when, 495
where, 503
wlf2log, 504
wlf2vcd, 506
wlfman, 507
wlfrecover, 514
write format, 515
write list, 518
write preferences, 519
write report, 520

write timing, 523
write transcript, 525
write tssi, 526
write wave, 528
commands formatTime, 190
comment characters in VSIM commands, 14
compatibility, of vendor libraries, 350
compiling
 range checking in VHDL, 342
 Verilog, 397
 VHDL, 331
 selected design units (-just eapbc), 336
 standard package (-s), 343
 VHDL-2008
 RE AL_VECTOR, 335
compressing files
 VCD files, 307, 318
concatenation
 directives, 41
 of signals, 40
conditional breakpoints, 495
conf igu r ations, simul ating, 429
configure command, 130
constants
 in case statements, 339
 values of, displaying, 153, 166
conversion
 radix, 241
coverage, automati c s ave, 429
coversto re, auto-s aved coverage, 429

— D —

dataset close command, 139
dataset current command, 142
dataset info command, 143
dataset list command, 144
dataset open command, 145
dataset rename command, 146
dataset restart command, 147
dataset save command, 148
dataset snapshot command, 149
datasets
 environment command, specifying with, 164
design lo ading, interr upting, 429
declarations, hiding i mplicit with explicit, 347

delay
 interconnect, 441
delete command, 152
deltas
 collapsing in WLF files, 453
dependencies, checking, 350
dependency errors, 401
describe command, 153
design units
 report of units simulated, 520
 Verilog
 adding to a library, 397
directories
 mapping libraries, 425
disablebp command, 154
dividers
 adding from command line, 78
divTime ccommand, 287
do command, 155
DO file
 executing, 155
DO files, 155
 breakpoints, executing at, 97
 forcing signals, nets, or registers, 184
 parameters
 passing, 155
 relative directories, 155
 shifting parameter values, 268
-dpiheader, vlog, 400
-dpiheader,vlog, 458
drivers command, 157
dump files, viewing in the simulator, 328
dumplog64 command, 159

— E —

echo command, 160
edit command, 161
enablebp command, 162
encoding command, 163
environment command, 164
environment variables
 reading into Verilo g code, 399
 specifying UNIX editor, 161
 state of, 233
 using in pathnames, 20

environment, displaying or changing pathname, 164

eqTime command, 287

errors

getting details about messages, 359

onerror command, 228

SDF, disabling, 446

event order

changing in Verilog, 398

examine command, 166

exit command, 173

extended identifiers, 20

— F —

file compression

VCD files, 307, 318

find command, 175

find connections command, 180

find infiles command, 181

find insource command, 182

fixed point radix, 244

floating point radix, 244

force

remove wire model, 443

force comma nd, 184

force command, 184

format file

List window, 515

Wave window, 515

formatTime command, 190, 287

— G —

gc configure command, 191

gc run command, 193

ge nerics

limitation on assigning composite types, 434

generics

assigning or overriding values with -g and -G, 434

examining generic values, 166

glitches

disabling generation

from command line, 456

global visibility

PLI/FLI shared objects, 435

gotolingk questa_sim_user

DPI Fi le Loading, 464

gteTime command, 287

gtTime command, 287

GUI_expression_format, 33

syntax, 34

— H —

hazards

-hazards argum ent to vsim, 459

-hazards argument to vlog, 402

help command, 194

history

of commands

shortcuts for reuse, 30

history command, 195

— I —

implicit operator, hiding with vcom -explicit, 347

interconnect de lays, 441

interconnect delays

annotat ing per Verilog 2001, 464

internal signals, adding to a VCD file, 303

interruptingdesignlo ading, 429

intToTime command, 287

— K —

keywords

enabling SystemVerilog keywords, 414

— L —

layout command, 196

LD_LIBRARY_PATH, disabling default internal s etting of, 442

li braries

Verilog, 439

libraries

dependencies, checking, 350

design libraries, creating, 393

listing contents, 350

refreshing library images, 412

vendor supplied, compatibility of, 350

lint-style checks, 404

List window

adding items to, 68

loading des igns, interrupti ng, 429

log command, 198

log file

 log command, 198

 nolog command, 220

 QuickSim II format, 504

 redirecting with -l, 438

 redirecting with -l, 438

 virtual log command, 379

 virtual nolog command, 382

lshift command, 201

lsublist command, 202

lteTime command, 287

ltTime command, 287

— M —

mc_scan_plusargs, PLI routine, 463

mem compare command, 203

mem display command, 204

mem list command, 207

mem load command, 208

mem save command, 212

mem search command, 215

memory window

 add memory command, 72

 adding items to, 72

memory, comparing contents, 203

memory, displaying contents, 204

memory, listing, 207

memory, loading contents, 208

memory, saving contents, 212

memory, searching for patterns, 215

messages

 echoing, 160

 getting more information, 359

 loading, disabling with -quiet, 411

 loading, disbling with -quiet, 342

-mfcu, 405

modelsim command, 218

mulTime command, 287

multi-source interconnect de lays, 441

— N —

name case sensitivity, VHDL vs. Verilog, 20

negative pulses

 driving an error state, 464

neqTime command, 287

nets

 drivers of, displaying, 157

 readers of, displaying, 252

 stimulus, 184

 values of

 examining, 166

-no_risefall_delaynets, 460

nolog command, 220

notepad command, 222

noview command, 223

nowhen command, 224

— O —

object_list_file, WLF files, 507

onbreak command, 225

onElabError command, 227

onerror command, 228

onfinish command, 230

optimizations

 disabling for VHDL designs, 341

 optimizing wlf files, 509

order of events

 changing in Verilog, 398

— P —

parameters

 using with DO files, 155

pathnames

 in VSIM commands, 15

 spaces in, 14

pause command, 231

PLI

 loading shared objects with global symbol visibility, 435

preference variables

 WildcardFilter, 23

printenv command, 232, 233

process report command, 234

projects

 override mapping for work directory with vcom, 346

 override mapping for work directory with vlog, 420

propagation, preventing X propagation, 442

pulse err or state, 464

pwd command, 238

— Q —

QuickSim II logfile format, 504
quietly command, 239
quit command, 240

— R —

Radix
 color, 245
 example, 246
radix
 display values in debug windows, 241
 of signals being examined, 70, 81, 169
 user defined, 244
radix command, 241
Radix define command
 setting radix color, 245, 246
radix define command, 244
 fixed point radix, 244
 floating point radix, 244
radix list command, 249
radix name command, 250
range checking
 disabling, 340
 enabling, 342
readers command, 252
RealToTime command, 287
record field selection, syntax, 16
refresh, dependency check errors, 401
refreshing library images, 412
report command, 253
reporting
 processes in the Process Window, 234
 variable settings, 27
resolution
 specifying with -t argument, 449
restart command, 255
resume command, 257
run command, 258
runStatus command, 261

— S —

scaleTime command, 287
scope resolution operator, 17
scope, setting region environment, 164
SDF
 annotation verbose mode, 447

controlling missing instance messages, 446
errors on loading, disabling, 446
warning messages, disabling, 446
search libraries, 439
searching
 binary signal values in the GUI, 43
 List window
 signal values, transitions, and names, 33
searchlog command, 263
see command, 266
setenv command, 267
shared objects
 loading with global symbol visibility, 435
shift command, 268
shortcuts
 command history, 30
 command line caveat, 29
show command, 269
signals
 alternative names in the Wave window (-label), 80
 attributes of, using in expressions, 34
 breakpoints, 495
 combining into a user-defined bus, 82
 drivers of, displaying, 157
 environment of, displaying, 164
 force time, specifying, 187
 log file, creating, 198
 pathnames in VSIM commands, 15
 radix
 specifying for examine, 70, 81, 169
 readers of, displaying, 252
 stimulus, 184
 values of
 examining, 166
simulating
 design unit, specifying, 429
simstats command, 270
simstatslist command, 272
simulating
 delays, specifying time units for, 27
 saving simulations, 198, 453
 stopping simulation in batch mode, 502
simulations

saving results, 148, 149
Simulator commands, 65
simulator resolution
 vsim -t argument, 449
simulator version, 452, 468
simultaneous events in Verilog
 changing order, 398
spaces in pathnames, 14
sparse memories
 listing with write report, 521
specify path delays, 464
stack down command, 274
stack frame command, 275
stack level command, 276
stack up command, 278
startup
 alternate to startup.do (vsim -do), 432
status command, 279
Std_logic
 mapping to binary radix, 43
stop command, 282
subTime command, 287
suppress command, 283
SystemC
 class and structure member naming syntax, 16
SystemVerilog
 multiple files in a compilation unit, 405
SystemVerilog
 enabling with -sv argument, 414
 scope resolution, 17

— T —

Tcl
 history shortcuts, 30
 variable
 in when commands, 499
TFMPC
 disabling warning, 461
timing
 disabling checks, 408
time
 absolute, using @, 27
 simulation time units, 27
time col lapsing, 453
Time commands, 287

time resolution
 setting
 with vsim command, 449
time, time units, simulation time, 27
timescale directive warning
 disabling, 461
timing
 disabling checks for entire design, 443
title, Main window, changing, 450
transcript
 redirecting with -l, 438
transcript command, 290
transcript file command, 291
transcript path command, 293
transcript sizelimit command, 294
TreeUpdate command, 518
TSCALE, disabling warning, 461
TSSI, 526
tssi2mti command, 298

— U —

-u, 419
undeclared nets, reporting an error, 404
unsetenv command, 301
user-defined bus, 82
User-defined radix, 244

— V —

-v, 419
v2k_int_delays, 464
validTime command, 287
values
 describe HDL items, 153
 examine HDL item values, 166
variable settings report, 27
variables
 describing, 153
 referencing in commands, 26
 value of
 changing from command line, 107
 examining, 166
vcd add command, 302
vcd checkpoint command, 305
vcd comment command, 306
vcd dumpports command, 307
vcd dumpportsall command, 310

-
- vcd dumpportsflush command, [311](#)
 - vcd dumpportslimit command, [312](#)
 - vcd dumpportsoff command, [314](#)
 - vcd dumpportson command, [315](#)
 - vcd file command, [316](#)
 - VCD files
 - adding items to the file, [302](#)
 - capturing port driver data, [307](#)
 - converting to WLF files, [328](#)
 - creating, [302](#)
 - dumping variable values, [305](#)
 - flushing the buffer contents, [321](#)
 - generating from WLF files, [506](#)
 - inserting comments, [306](#)
 - internal signals, adding, [303](#)
 - specifying maximum file size, [323](#)
 - specifying name of, [318](#)
 - specifying the file name, [316](#)
 - state mapping, [316, 318](#)
 - turn off VCD dumping, [325](#)
 - turn on VCD dumping, [326](#)
 - viewing files from another tool, [328](#)
 - vcd files command, [318](#)
 - vcd flush command, [321](#)
 - vcd limit command, [323](#)
 - vcd off command, [325](#)
 - vcd on command, [326](#)
 - vcd2wlf command, [328](#)
 - vcom command, [331](#)
 - vdel command, [348](#)
 - vdir command, [350](#)
 - vector elements, initializing, [107](#)
 - vencrypt command, [353](#)
 - vendor libraries, compatibility of, [350](#)
 - Verilog
 - capturing port driver data with -dumpports, [316](#)
 - verror command, [359](#)
 - version
 - obtaining with vsim command, [452](#)
 - obtaining with vsimcommands, [468](#)
 - vgencomp command, [361](#)
 - VHDL
 - binding, ignore default, [335](#)
 - field naming syntax, [16](#)
 - VHDL-2008
 - package STANDARD
 - REAL_VECTOR, [335](#)
 - vhencrypt command, [363](#)
 - viewing
 - wav eforms, [453](#)
 - virtual count commands, [370](#)
 - virtual define command, [371](#)
 - virtual delete command, [372](#)
 - virtual describe command, [373](#)
 - virtual expand commands, [374](#)
 - virtual function command, [375](#)
 - virtual hide command, [378](#)
 - virtual log command, [379](#)
 - virtual nohide command, [381](#)
 - virtual nolog command, [382](#)
 - virtual region command, [384](#)
 - virtual save command, [385](#)
 - virtual show command, [386](#)
 - virtual signal command, [387](#)
 - vlib command, [393](#)
 - vlog
 - multiple file compilation, [405](#)
 - vlog command, [397](#)
 - vmake command, [423](#)
 - vmap command, [425](#)
 - vsim, [427](#)
 - disabling internal setting of LD_LIBRARY_PATH, [442](#)
 - W —
 - warnings
 - SDF, disabling, [446](#)
 - suppressing VCOM warning messages, [341, 409](#)
 - suppressing VLOG warning messages, [409](#)
 - suppressing VSIM warning messages, [461](#)
 - watch window
 - add watch command, [76](#)
 - watching signal values, [76](#)
 - wave commands, [472](#)
 - wave create command, [477](#)
 - wave cursor commands, [472](#)
 - wave edit command, [483](#)
 - wave export command, [486](#)
 - wave import command, [488](#)

wave log format (WLF) file, 453
 of binary signal values, 198

wave modify command, 489

wave sort command, 494

Wave window
 adding items to, 77

WaveActivateNextPane command, 518

waveform editor
 creating waves, 477
 editing commands, 483
 importing vcd stimulus file, 488
 modifying existing waves, 489
 saving waves, 486

waveform logfile
 log command, 198

waveforms
 saving and viewing, 198

WaveRestoreCursors command, 518

WaveRestoreZoom command, 518

when command, 495

when statement
 time-based breakpoints, 502

where command, 503

wildcard characters
 for pattern matching in simulator
 commands, 22

WildcardFilter Preference Variable, 23

windows

- List window
 - output file, 518
 - saving the format of, 515
- Wave window
 - path elements, changing, 133

WLF files
 collapsing deltas, 453
 collapsing time steps, 453
 converting to VCD, 506
 creating from VCD, 328
 indexing, 509
 limiting size, 454
 log command, 198
 merging, 509
 optimizing, 509
 repairing, 514
 saving, 148, 149

specifying name, 453
wlfman command, 507

wlf2log command, 504

wlf2vcd command, 506

wlfman command, 507

wlfrecover com mand, 514

write format command, 515

write list command, 518

write preferences command, 519

write report command, 520

write timing command, 523

write transcript command, 525

write tssi command, 526

write wave command, 528

— X —

Xpropagation
 disabling for entire design, 442

— Y —

-y, 420

— Z —

zoom
 wave window
 returning current range, 475

End-User License Agreement with EDA Software Supplemental Terms

Use of software (including any updates) and/or hardware is subject to the End-User License Agreement together with the Mentor Graphics EDA Software Supplement Terms. You can view and print a copy of this agreement at:

mentor.com/eula

