ModelSim® Reference Manual

Software Version 6.3g

May 2008

© 1991-2008 Mentor Graphics Corporation
All rights reserved.

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

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

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

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

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

RESTRICTED RIGHTS LEGEND 03/97

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

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

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

Table of Contents

List of Examples

List of Figures

List of Tables

Chapter 1
Syntax and Conventions

Documentation Conventions

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

Table 1-1. Conventions for Command Syntax

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

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

add wave {vector1[4:0]}

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

Note

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

File and Directory Pathnames

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

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

or

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

Design Object Names

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

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

Object Name Syntax

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

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

where

- **datasetName** — is the logical name of the WLF file in which the object exists. The currently active simulation is the “sim” dataset. Any loaded WLF file is referred to by the logical name specified when the WLF file was loaded. Refer to the chapter “Recording Simulation Results With Datasets” in the User’s Manual for more information.
- **datasetSeparator** — is the character used to terminate the dataset name. The default is ’:’, though a different character (other than ’\’) may be specified as the dataset separator via the DatasetSeparator variable in the *modelsim.ini* file. The default is ':'. This character must be different than the pathSeparator character.
- **pathSeparator** — is the character used to separate hierarchical object names. Normally, '/' is used for VHDL and '.' is used for Verilog, although other characters (except '\') may be specified via the PathSeparator variable in the *modelsim.ini* file. This character must be different than the datasetSeparator. Neither '.' or '/' can be used when referring to the contents of a SystemVerilog package or class.

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

SystemVerilog Scope Resolution Operator

SystemVerilog offers the scope resolution operator '::' 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 examine examples access data from the class *packet*.

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

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

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

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

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

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

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

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

When referring to the contents of a package or class, you cannot use the standard path separators '.' or '/'.

Specifying Names

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

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

A relative name does not start with a path separator and may or may not include a dataset name or a hierarchical path (e.g., *u1/data* or *view:clk*). A relative name is relative to the current context in the current or specified dataset.

A fully-rooted name starts with a path separator and includes a hierarchical path to an object (e.g., */top/u1/clk*).There is a special case of a fully-rooted name where the top-level design unit name can be unspecified (e.g., */u1/clk*). In this case, the first top-level instance in the design is assumed.

An absolute name is an exactly specified hierarchical name containing a dataset name and a fully rooted name (e.g., *sim:/top/u1/clk*).

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

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

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

Table 1-2. Examples of Object Names

Object Name	Description
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
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 Escaping Brackets and Spaces in Array Slices for more information
{mysignal[31:0]}	specifies a slice of an array object in Verilog; see Escaping Brackets and Spaces in Array Slices for more information
record_sig.field	specifies a field of a record

Escaping Brackets and Spaces in Array Slices

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

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

For complete details on Tcl syntax, refer to Tcl Command Syntax.

Further Details

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

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

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

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

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

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

Environment Variables and Pathnames

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

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

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

Note

Environment variable expansion *does not* occur in files that are referenced via the **-f** argument to **vcom**, **vlog**, or **vsim**.

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

Wildcard characters can be used in HDL object names in some simulator commands. Table 1-3 shows the conventions for allowable 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

Filtering Wildcard Matching for Certain Commands

By default certain commands do not add all objects that match a wildcard pattern. For example, the **add wave** command doesn't add VHDL variables or Verilog memories by default because this can use a lot of RAM.

For example, "add log -r *" will not log process variables, even if you change the WildcardFilter preference variable, because the filter only applies to variables in packages and design units. To use wildcards for other variables, you must specify an explicit path to the process in the log command, such as "add log /proc_test/p1/*".

Note

A wildcard character will never match a path separator. For example, */dut/** will match */dut/siga* and */dut/clk*. However, */dut** won't match either of those.

WildcardFilter Preference Variable

The *WildcardFilter* preference variable allows you to specify the object types to exclude when performing wildcard matches. The settings will be applied only to variables in packages and design units. See "Filtering Wildcard Matching for Certain Commands" for more information.

You can view the current settings by entering the following at the command prompt:

```
> set WildcardFilter
```

which, returns a space separated list of the current values. The default setting returns:

```
> set WildcardFilter
# Variable Constant Generic Parameter SpecParam Memory CellInternal
```

You can change the settings by specifying a space-separated list of values enclosed in double-quotes (").

For example:

```
set WildcardFilter "signal constant compare net reg"
```

Table 1-4 lists all the legal values for WildcardFilter.

Table 1-4. WildcardFilter Values

Alias	Memory	Reg
CellInternal[1]	NamedEvent	Signal
Compare	Net	SpecParam
Constant	None[2]	Time
Generic	Parameter	Variable
Integer	Real	

1. Applies to signals in cells, where a cell is defined as 1) a module within a ‘celldefine, 2) a Verilog module referenced as a library via the -v or -y compiler options, or 3) a module containing a specify block.
2. This value disables all filtering.

The WildcardFilter variable applies to the commands add dataflow, add list, add memory, add watch, add wave, find, and log.

Simulator Variables

ModelSim variables can be referenced in simulator commands by preceding the name of the variable with the dollar sign ($) character. ModelSim uses global variables for simulator state variables, simulator control variables, simulator preference variables, and user-defined variables. Refer to “Simulator State Variables” in the User’s Manual for more information on variables.

The report command returns a list of current settings for either the simulator state or simulator control variables.

Simulation Time Units

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

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

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

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

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

Argument Files

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

```
vlog   vcom   vsim
```

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

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

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

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

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

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

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

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

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

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

- Newline Character — you can specify arguments on separate lines in the argument file, with the newline characters treated as space characters. There is no need to put '\' at the end of each line.

- Comments — Comments within the argument files follow these rules:
  - All text in a line beginning with // to its end is treated as a comment.
  - All text bracketed by /* … */ is treated as a comment.

Command Shortcuts

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

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

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

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

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

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

Command History Shortcuts

You can review simulator command history or rerun previous commands by using keyboard shortcuts at the ModelSim/VSIM prompt. Table 1-5 contains a list of these shortcuts.

Table 1-5. Keyboard Shortcuts for Command History

Shortcut	Description
!!	repeats the last command
!n	repeats command number n; n is the VSIM prompt number (e.g., for this prompt: VSIM 12>, n =12)
!abc	repeats the most recent command starting with "abc"
^xyz^ab^	replaces "xyz" in the last command with "ab"
up and down arrows	scrolls through the command history with the keyboard arrows

Table 1-5. Keyboard Shortcuts for Command History

Shortcut	Description
click on prompt	left-click once on a previous ModelSim or VSIM prompt in the transcript to copy the command typed at that prompt to the active cursor
his or history	shows the last few commands (up to 50 are kept)

Numbering Conventions

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

VHDL Numbering Conventions

There are two types of VHDL number styles:

VHDL Style 1

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

Table 1-6. VHDL Number Conventions: Style 1

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

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

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

VHDL Style 2

```
base "value"
```

Table 1-7. VHDL Number Conventions: Style 2

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

For example:

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

Searching for VHDL Arrays in the Wave and List Windows

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

Verilog Numbering Conventions

Verilog numbers are expressed in the style:

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

Table 1-8. Verilog Number Conventions

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

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

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

GUI_expression_format

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

configure, examine, searchlog, virtual function, virtual signal

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 1-9. Constants Supported for GUI Expresssions

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

Array Constants, Expressed in Any of the Following Formats

Table 1-10. Array Constants Supported for GUI Expresssions

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 1-11. Variables Supported for GUI Expresssions

Variable	Type
Name of a signal	The name may be a simple name, a VHDL or Verilog style extended identifier, or a VHDL or Verilog style path. The signal must be one of the following types: -- VHDL signal of type INTEGER, REAL, or TIME -- VHDL signal of type std_logic or bit -- VHDL signal of type user-defined enumeration -- Verilog net, Verilog register, Verilog integer, or Verilog real
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 1-12. Array Variables Supported for GUI Expresssions

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

Signal attributes

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

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

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

See Examples of Expression Syntax below for further details on 'delayed and 'hasX.

Operators

Table 1-13. Operators Supported for GUI Expresssions

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

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

Note

Arithmetic operators use the std_logic_arith package.

Casting

Table 1-14. Casting Conversions Supported for GUI Expresssions

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

Examples of Expression Syntax

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Signal and Subelement Naming Conventions

ModelSim supports naming conventions for VHDL and Verilog signal pathnames, VHDL array indexing, Verilog bit selection, VHDL subrange specification, and Verilog part selection.

Examples in Verilog and VHDL syntax:

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

Grouping and Precedence

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

Concatenation of Signals or Subelements

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

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

Concatenation Syntax for VHDL

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

Concatenation Syntax for Verilog

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

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

Concatenation Directives

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

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

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

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

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

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

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

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

The concat_noflatten directive groups signals together without merging them into one big array. The signals become elements of a record and retain their original names. When expanded, the new signal looks just like a group of signals. The directive can be used hierarchically with no limits on depth.

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

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

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

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

Examples of Concatenation

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

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

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

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

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

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

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

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

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

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

Record Field Members

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

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

Examples:

```
vhdlsig.field1
vhdlsig.field1.subfield1
vhdlsig.(5).field3
vhdlsig.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 testbenches.

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

Table 1-15. 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 1-16. 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

The commands here are entered either in macro files or on the command line of the Main window. Some commands are automatically entered on the command line when you use the ModelSim graphical user interface.

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

The following table provides a brief description of each ModelSim command. Command details, arguments, and examples can be selecting the links in the Command name column.

Table 2-1. Supported Commands

Command name	Action
abort	halts the execution of a macro file interrupted by a breakpoint or error
add dataflow	adds the specified object to the Dataflow window
add list	lists VHDL signals and variables, and Verilog nets and registers, and their values in the List window
add log	also known as the **log** command; see log
add memory	opens the specified memory in the MDI frame of the Main window
add watch	adds signals or variables to the Watch window
add wave	adds VHDL signals and variables, and Verilog nets and registers to the Wave window
add_cmdhelp	adds an entry to the command-line help; use the help command to display the help text
alias	creates a new Tcl procedure that evaluates the specified commands
batch_mode	returns a 1 if ModelSim is operating in batch mode, otherwise returns a 0
bd	deletes a breakpoint
bookmark add wave	adds a bookmark to the specified Wave window
bookmark delete wave	deletes bookmarks from the specified Wave window
bookmark goto wave	zooms and scrolls a Wave window using the specified bookmark

Table 2-1. Supported Commands (cont.)

Command name	Action
bookmark list wave	displays a list of available bookmarks
bp	sets a breakpoint
change	modifies the value of a VHDL variable or Verilog register variable
configure	invokes the List or Wave widget configure command for the current default List or Wave window
dataset alias	assigns an additional name to a dataset
dataset clear	clears the current simulation WLF file
dataset close	closes a dataset
dataset config	configures WLF file settings after dataset is open
dataset info	reports information about the specified dataset
dataset list	lists the open dataset(s)
dataset open	opens a dataset and references it by a logical name
dataset rename	changes the logical name of an opened dataset
dataset restart	unloads specified or current dataset
dataset save	saves data from the current WLF file to a specified file
dataset snapshot	saves data from the current WLF file at a specified interval
delete	removes objects from either the List or Wave window
describe	displays information about the specified HDL object
disablebp	turns off breakpoints and when commands
do	executes commands contained in a macro file
drivers	displays in the Main window the current value and scheduled future values for all the drivers of a specified VHDL signal or Verilog net
dumplog64	dumps the contents of the *vsim.wlf* file in a readable format
echo	displays a specified message in the Main window
edit	invokes the editor specified by the EDITOR environment variable
enablebp	turns on breakpoints and when commands turned off by the disablebp command
environment	displays or changes the current dataset and region environment
examine	examines one or more objects, and displays current values (or the values at a specified previous time) in the Main window

Table 2-1. Supported Commands (cont.)

Command name	Action
exit	exits the simulator and the ModelSim application
find	displays the full pathnames of all objects in the design whose names match the name specification you provide
find infiles	searches the specified files and prints to the Transcript pane those lines from the files that match the specified pattern.
find insource	searches all source files related to the current design and prints to the Transcript pane those lines from the files that match the specified pattern.
formatTime	global format control for all time values displayed in the GUI
force	applies stimulus to VHDL signals and Verilog nets
help	displays in the Main window a brief description and syntax for the specified command
history	lists the commands executed during the current session
layout	save or load custom GUI layouts
log	creates a wave log format (WLF) file containing simulation data for all objects whose names match the provided specifications
lshift	takes a Tcl list as an argument and shifts it in-place one place to the left, eliminating the left-most element
lsublist	returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern
mem compare	compares the selected memory to a reference memory or file
mem display	displays the memory contents of a selected instance to the screen
mem list	displays a flattened list of all memory instances in the current or specified context after a design has been elaborated
mem load	updates the simulation memory contents of a specified instance
mem save	saves the contents of a memory instance to a file in any of the supported formats: Verilog binary, Verilog hex, and MTI memory pattern data
mem search	finds and prints to the screen the first occurring match of a specified memory pattern in the specified memory instance
modelsim	starts the ModelSim GUI without prompting you to load a design; valid only for Windows platforms
noforce	removes the effect of any active force commands on the selected object
nolog	suspends writing of data to the WLF file for the specified signals

Table 2-1. Supported Commands (cont.)

Command name	Action
notepad	opens a simple text editor
noview	closes a window or set of windows in the ModelSim GUI
nowhen	deactivates selected when commands
onbreak	specifies command(s) to be executed when running a macro that encounters a breakpoint in the source code; in effect only during a run command
onElabError	specifies one or more commands to be executed when an error is encountered during elaboration; in effect only during a vsim command
onerror	specifies one or more commands to be executed when a Tcl command in a dofile encounters an error; not dependent on a run command
pause	interrupts the execution of a macro
precision	determines how real numbers display in the GUI
printenv	echoes to the Main window the current names and values of all environment variables
project	performs common operations on new projects
pwd	displays the current directory path in the Main window
quietly	turns off transcript echoing for the specified command
quit	exits the simulator
radix	specifies the default radix to be used
radix define	creates or modifies a user-defined radix
radix names	returns a list of currently defined radix names
radix list	returns the complete definition of a radix
radix delete	removes the radix definition from the named radix
report	displays the value of all simulator control variables, or the value of any simulator state variables relevant to the current simulation
restart	reloads the current dataset if the current dataset is not the active simulation ("sim") and resets the simulation time to zero, in effect acting just like a restart of a simulation
resume	resumes execution of a macro file after a pause command or a breakpoint
run	advances the simulation by the specified number of timesteps

Table 2-1. Supported Commands (cont.)

Command name	Action
searchlog	searches one or more of the currently open logfiles for a specified condition
setenv	sets an environment variable
shift	shifts macro parameter values down one place
show	lists objects and subregions visible from the current environment
simstats	reports performance-related statistics about active simulations
status	lists all currently interrupted macros
step	steps to the next HDL statement
stop	stops simulation in batch files; used with the when command
suppress	prevents the specified message(s) from displaying
tb	displays a stack trace for the current process in the Transcript pane
transcript	controls echoing of commands executed in a macro file; also works at top level in batch mode
transcript file	sets or queries the pathname for the transcript file
tssi2mti	converts a vector file in Technology Standard Events Format (TSSI) into a sequence of force and run commands
unsetenv	deletes an environment variable
vcd add	adds the specified objects to the VCD file
vcd checkpoint	dumps the current values of all VCD variables to the VCD file
vcd comment	inserts the specified comment in the VCD file
vcd dumpports	creates a VCD file that captures port driver data
vcd dumpportsall	creates a checkpoint in the VCD file that shows the current values of all selected ports
vcd dumpportsflush	flushes the VCD buffer to the VCD file
vcd dumpportslimit	specifies the maximum size of the VCD file
vcd dumpportsoff	turns off VCD dumping and records all dumped port values as x
vcd dumpportson	turns on VCD dumping and records the current values of all selected ports
vcd file	specifies the filename and state mapping for the VCD file created by a vcd add command

Table 2-1. Supported Commands (cont.)

Command name	Action
vcd files	specifies filenames and state mapping for the VCD files created by the vcd add command; supports multiple VCD files
vcd flush	flushes the contents of the VCD file buffer to the VCD file
vcd limit	specifies the maximum size of the VCD file
vcd off	turns off VCD dumping and records all VCD variable values as x
vcd on	turns on VCD dumping and records the current values of all VCD variables
vcd2wlf	translates VCD files into WLF files
vcom	compiles VHDL design units
vdel	deletes a design unit from a specified library
vdir	lists the contents of a design library
vencrypt	encrypts Verilog code contained within encryption envelopes
verror	prints a detailed description of a message number
vgencomp	writes a Verilog module's equivalent VHDL component declaration to standard output
view	opens a ModelSim window and brings it to the front of the display
virtual count	counts the number of currently defined virtuals that were not read in using a macro file
virtual define	prints the definition of a virtual signal or function in the form of a command that can be used to re-create the object
virtual delete	removes the matching virtuals
virtual describe	prints a complete description of the data type of one or more virtual signals
virtual expand	produces a list of all the non-virtual objects contained in the virtual signal(s)
virtual function	creates a new signal that consists of logical operations on existing signals and simulation time
virtual hide	causes the specified real or virtual signals to not be displayed in the Objects window
virtual log	causes the sim-mode dependent signals of the specified virtual signals to be logged by the simulator
virtual nohide	redisplays a virtual previously hidden with **virtual hide**
virtual nolog	stops the logging of the specified virtual signals

Table 2-1. Supported Commands (cont.)

Command name	Action
virtual region	creates a new user-defined design hierarchy region
virtual save	saves the definitions of virtuals to a file
virtual show	lists the full path names of all the virtuals explicitly defined
virtual signal	creates a new signal that consists of concatenations of signals and subelements
virtual type	creates a new enumerated type
vlib	creates a design library
vlog	compiles Verilog design units and SystemVerilog extensions
vmake	creates a makefile that can be used to reconstruct the specified library
vmap	defines a mapping between a logical library name and a directory
vsim	loads a new design into the simulator
vsim<info>	returns information about the current vsim executable
vsim_break	stop the current simulation before completion
vsource	specifies an alternative file to use for the current source file
wave	commands for manipulating cursors, for zooming, and for adjusting the wave display view in the Wave window
when	instructs ModelSim to perform actions when the specified conditions are met
where	displays information about the system environment
wlf2log	translates a ModelSim WLF file to a QuickSim II logfile
wlf2vcd	translates a ModelSim WLF file to a VCD file
wlfman	outputs information about or a new WLF file from an existing WLF file
wlfrecover	attempts to repair an incomplete WLF file
write format	records the names and display options in a file of the objects currently being displayed in the List or Wave window
write list	records the contents of the List window in a list output file
write preferences	saves the current GUI preference settings to a Tcl preference file
write report	prints a summary of the design being simulated
write timing	prints timing information about the specified instance

Table 2-1. Supported Commands (cont.)

Command name	Action
write transcript	writes the contents of the Main window transcript to the specified file
write tssi	records the contents of the List window in a “TSSI format” file
write wave	records the contents of the Wave window in PostScript format

abort

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

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

Syntax

abort [<n> | all]

Arguments

- <n>

  (optional) An integer, greater than 0, that specifies the number of nested macro levels to abort, where the default value of is 1.

- all

  (optional) A literal that instructs the tool to abort all levels of nested macros.

See also

onbreak onElabError onerror

add dataflow

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

Syntax

add dataflow **<object>** ... { [-in] [-out] [-inout] | [-ports] } [-internal] [-nofilter] [-recursive]

- **<object>** ...

  (required) A string, which is repeatable in a space separated list, that specifies a process, signal, net, or register that you want to add to the Dataflow window, where wildcards are allowed. Refer to the section "Filtering Wildcard Matching for Certain Commands" for wildcard usage as it pertains to the add commands.

- -in

  (optional) A literal that specifies to add ports of mode IN.

- -inout

  (optional) A literal that specifies to add ports of mode INOUT.

- -internal

  (optional) A literal that specifies to add internal (non-port) objects.

- -nofilter

  (optional) A literal that 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.

- -out

  (optional) A literal that specifies to add ports of mode OUT.

- -ports

  (optional) A literal that specifies to add all ports. This switch has the same effect as specifying **-in**, **-out**, and **-inout** together.

- -recursive

  (optional) A literal that 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.

See also

Dataflow Window
WildcardFilter
Preference Variable

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 *
```

add list

The **add list** command adds the following objects and their values to 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, **add list** displays all objects in the selected region with names matching the object name specification.

See "Filtering Wildcard Matching for Certain Commands" for wildcard usage as it pertains to the add commands.

Syntax

add list [-radix <type> | -<radix_type>] [-width <integer>] [-allowconstants] [-depth <level>] {[-in] [-inout] [-out] | [-ports]} [-internal] [-label <name>] [-nodelta] [-trigger | -notrigger] [-recursive] {**<object> ...** | **<object_name> {sig ...}**}

Arguments

- **<object> ...**

  (required) A string, which is repeatable in a space-separated list, that specifies the name(s) of the object to be listed, where wildcards are allowed. Refer to the section "Filtering Wildcard Matching for Certain Commands" 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) A group of arguments, enclosed in braces ({ }), that creates a user-defined bus with the specified object name containing the specified signals (sig) concatenated within the user-defined bus.

  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

  For use with wildcard searches. (optional) A switch that 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) A switch and argument pair that restricts a recursive search, as specified with **-recursive**, to a certain level of hierarchy.

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

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

- -in

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

- -inout

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

- -internal

  For use with wildcard searches. (optional) A switch that 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) A switch and argument pair that 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 whitespace.

  This alternative name is not valid in a force or examine command.

- -nodelta

  (optional) A switch that specifies that the delta column not be displayed when adding signals to the List window. Identical to **configure list -delta none**.

- -out

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

- -ports

  For use with wildcard searches. (optional) A switch that 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.

- -radix <type> | -<radix_type>

  (optional) A choice between switches that specify the radix for the objects that follow in the command. Valid entries (or any unique abbreviations) are:

  -radix binary -binary

-radix ascii	-ascii
-radix unsigned	-unsigned
-radix decimal	-decimal
-radix octal	-octal
-radix hex	-hex
-radix symbolic	-symbolic
-radix time	-time
-radix default	-default

If no radix is specified for an enumerated type, the default representation is used.

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

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.

- -recursive

  For use with wildcard searches. (optional) A switch that 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) A choice of switches that specify whether objects should be updated in the List window when the objects change value.

- -width <integer>

  (optional) A switch and argument pair that specifies 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
```

See also

add wave | log | Extended Identifiers | WildcardFilter Preference Variable

add memory

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

See "Filtering Wildcard Matching for Certain Commands" for wildcard usage as it pertains to the add commands.

Syntax

add memory [-addressradix {decimal | hex}] [-dataradix <type>] [-wordsperline <num>] **<object_name> ...**

Arguments

- -addressradix {decimal | hex}

  (optional) A switch and argument pair that 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 hexidecimal. You can abbreviate this to "h".

- -dataradix <radix_type>

  (optional) A switch and argument pair that specifies the data radix for the memory display.

  If you do not specify this switch, the command uses the global default radix.

  <type> — Valid entries (or any unique abbreviations) are:

  -binary

  -unsigned

  -decimal

  -octal

  -hex

  -symbolic

  -default

  If you do not specify a radix is specified for an enumerated type, the command uses the symbolic representation.

  You can change the default radix for the current simulation using the radix command. You can change the default radix permanently by editing the DefaultRadix variable in the *modelsim.ini* file. Changing the default radix does not change the radix of the currently-displayed memory. Use the **add memory** command to re-add the memory with the desired radix, or change the display radix from the Memory window Properties dialog.

- -wordsperline <num>

  (optional) A switch and argument pair that specifies how many words are displayed on each line in the memory window.

  By default, the information displayed will wrap based on the width of the window.

- **<object_name> ...**

  (required) A string, which is repeatable in a space-separated list, that specifies the hierarchical path of the memory to be displayed.

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

See also

Memory Panes
WildcardFilter
Preference Variable

add watch

The **add watch** command adds signals and variables to the Watch window in the Main window. See "Filtering Wildcard Matching for Certain Commands" for wildcard usage as it pertains to the add commands.

Syntax

add watch **<object_name> ...** [-radix <type>]

Arguments

- **<object_name> ...**

  (required) A string, which is repeatable in a space-separated list, that specifies the name of the object to be added.

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

  Variables must be preceded by the process name. For example,

  ```
  add watch myproc/int1
  ```

- -radix <type>

  (optional) A switch and argument pair that specifies a user-defined radix.

  If you do not specify this switch, the command uses the global default radix.

  <type> — Valid entries (or any unique abbreviations) are:

  -binary

  -ascii

  -unsigned

  -decimal

  -octal

  -hex

  -symbolic

  -time

  -default

See also

Watch Pane

WildcardFilter Preference Variable

add wave

The **add wave** command adds the following objects to the Wave window:

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

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

See "Filtering Wildcard Matching for Certain Commands" for wildcard usage as it pertains to the add commands.

Syntax

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

Arguments

- -allowconstants

  For use with wildcard searches. (optional) A switch that specifies that constants matching the wildcard search should be added to the Wave window.

  By default, constants are ignored because they do not change.

- -clampanalog {0 | 1}

  (optional) A switch and argument pair that 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) A switch and argument pair that 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) A switch and argument pair that restricts a recursive search, as specified with **-recursive** to a specified 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.

- -divider [<divider_name> ...]

  (optional) A switch and argument pair that adds a divider to the Wave window.

  <divider_name> ... — A string, which is repeatable in a space separated list, that specifies the name of the divider, which appears in the pathnames column.

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

  You cannot begin a name with a hyphen (-).

  You can begin a name with a space, but you must enclose the name within quotes (") or braces ({ })

  If you do not specify this argument, the command inserts an unnamed divider.

- -expand <signal_name>

  (optional) A switch and argument pair that instructs the command to expand a compound signal immediately, but only one level down.

  <signal_name> — a string that specifies the name of the signal. This string can include wildcards.

- -<format>

  (optional) A switch that specifies the display format of the objects. The switches are:

  -literal — Literal waveforms are displayed as a box containing the object value.

  -logic — Logic signals may be U, X, 0, 1, Z, W, L, H, or '-'.

  -analog-step — Analog-step changes to the new time before plotting the new Y.

  -analog-interpolated — Analog-interpolated draws a diagonal line.

  -analog-backstep — Analog-backstep plots the new Y before moving to the new time.

  The Y-axis range of analog signals is bounded by -max and -min switches. Refer to "Formatting the Wave Window" for more information.

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

  (optional) A switch and argument group that creates a wave group with the specified group_name.

  <group_name> — a string that specifies the name of the group. You must enclose this argument in quotes (") or braces ({ }) if it contains any white space.

<sig_name> ... — a string, which is repeatable in a space separated list, that specifies the signals to add to the group. This command creates an empty group if you do not specify any signal names.

- -height <pixels>

  (optional) A switch and argument pair that specifies the height, in pixels, of the waveform.

- -in

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

- -inout

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

- -internal

  For use with wildcard searches. (optional) A switch that 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) A switch and argument pair that 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) A switch and argument pair that specifies the maximum Y-axis data value to be displayed for an analog waveform. Used in conjunction with the -min switch; the value you specify for -max must be greater than the value you specify for -min.

- -min <real_num>

  (optional) A switch and argument pair that specifies the minimum Y-axis data value to be displayed for an analog waveform. Used in conjunction with the -max switch; the value you specify for -min must be less than the value you specify for -max.

  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 you can still use the -offset and -scale switches, the -max and -min switches are provided as an easier way to define upper and lower limits of an analog waveform.

- -noupdate

  (optional) A switch that prevents the Wave window from updating when a series of add wave commands are executed in series.

- **<object_name> ...**

  (required) A string, which is repeatable in a space separated list, that specifies the names of objects to be included in the Wave window. Wildcard characters are allowed. Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.

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

  ```
  add wave myproc/int1
  ```

- {<object_name> {sig ...}}

  (required) A group of arguments, enclosed in braces ({ }), that creates a user-defined bus with the specified object name containing the specified signals (sig) concatenated within the user-defined bus.

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

Note

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

- -out

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

- -ports

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

- -position <location>

  (optional) A switch and argument pair that 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 the end of the list of signals.

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

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

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

- -radix <type> | -<radix_type>

  (optional) A choice between switches that specify the radix for the objects that follow in the command. Valid entries (or any unique abbreviations) are:

-radix binary	-binary
-radix ascii	-ascii
-radix unsigned	-unsigned
-radix decimal	-decimal
-radix octal	-octal
-radix hex	-hex
-radix symbolic	-symbolic
-radix time	-time
-radix default	-default

  If no radix is specified for an enumerated type, the default representation is used.

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

  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.

- -recursive

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

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

- -time

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

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}}
```

- Wave all objects in the region.

```
add wave *
```

- Wave all 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 {}
  ```

See also

add list | log | Extended Identifiers | WildcardFilter Preference Variable | Concatenation Directives

add_cmdhelp

The **add_cmdhelp** command adds the specified command name, description, and command arguments to the command-line help. You can then access the information using the help command.

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

Syntax

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

Arguments

- {<command_name>}

  (required) A string, enclosed in braces ({ }), that specifies the command name that will be entered as an argument to the **help** command. The command_name must not interfere with an already existing command_name.

- {<command_description>}

  (required) A string, enclosed in braces ({ }), that specifies a description of the command.

- {<command_arguments>}

  (required) A space-separated list of arguments, enclosed in braces ({ }), for the command. If the command doesn't have any arguments, enter {}.

Examples

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

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

VSIM> help date
Displays date and time.
Usage: date
```

- Add the change date command.

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

VSIM> help change date
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

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

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

Syntax

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

Arguments

- <name>

  (optional) A string that specifies the new procedure name to be used when invoking the commands.

- "<cmds>"

  (optional) A string, enclosed in quotes ("), that specifies the command or commands to be evaluated when the alias is invoked. You must separate multiple commands with a semicolon (;).

Examples

- List all aliases currently defined.

  ```
  alias
  ```

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

  ```
  alias <name>
  ```

- Create a Tcl procedure, "myquit", that when executed, writes the contents of the List window to the file *mylist.save* by invoking write list, and quits ModelSim by invoking quit.

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

batch_mode

The **batch_mode** command returns a 1 if ModelSim is operating in batch mode, otherwise it returns a 0. It is typically used as a condition in an if statement.

Syntax

batch_mode

Arguments

None

Examples

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

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

See also

"Modes of Operation"

bd

The **bd** command deletes a breakpoint. You must specify a filename and line number or a specific breakpoint id#. You may specify multiple filename/line number pairs and id#s.

Syntax

bd {{<filename> <line_number>} | <id_number>} ...

Arguments

- <filename>

  (required) A string that specifies the name of the source file in which the breakpoint is to be deleted. The filename must match the one used previously to set the breakpoint, including whether you used a full pathname or a relative name.

- <line_number>

  (required) A string that specifies the line number of the breakpoint to be deleted.

- <id_number>

  (required) A string that specifies the identification number of the breakpoint to be deleted.

  If you are deleting a C breakpoint, the identification number will have a "c" prefix.

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 id# 6 and the breakpoint at line 234 in the source file named *alu.vhd*.

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

See also

bp onbreak

bookmark add wave

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

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

Syntax

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

Arguments

- <label>

  (required) A string that specifies the name for the bookmark.

- [<range_start> [<unit>]] <range_end> [<unit>]

  (optional) A group of strings that specify the beginning and end points of the zoom range. You must enclose these arguments within braces ({}) or quotation marks ("").

  If you do not specify the <range_start> argument the bookmark will begin with zero.

  The tool uses your current time unit if you do not specify <unit>.

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

  ```
  {{100 ns} {10000 ns}}
  {10000}
  ```

- <topindex>

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

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
  ```

See also

bookmark delete wave | bookmark goto wave | bookmark list wave | write format

bookmark delete wave

The **bookmark delete wave** command deletes bookmarks from the specified Wave window.

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

Syntax

bookmark delete wave {<label> | -all]

Arguments

- <label>

  (required) A string that specifies the name of the bookmark to delete. You must specify this argument unless you specify -all.

- -all

  (optional) A switch that specifies that all bookmarks in the window be deleted.

Examples

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

  ```
  bookmark delete wave foo
  ```

- Delete all bookmarks from the Wave window named "wave1".

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

See also

bookmark add wave | bookmark goto wave | bookmark list wave | write format

bookmark goto wave

The **bookmark goto wave** command zooms and scrolls a Wave window using the specified bookmark.

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

Syntax

bookmark goto wave <label>

Arguments

- <label>

  (required) A string that specifies the bookmark to go to.

See also

bookmark add wave bookmark delete wave bookmark list wave write format

bookmark list wave

The **bookmark list wave** command displays a list of available bookmarks in the Transcript pane.

Syntax

bookmark list wave

See also

bookmark add wave bookmark delete wave bookmark goto wave write format

bp

The **bp** or breakpoint command either sets a file-line breakpoint or returns a list of currently set breakpoints.

Syntax

Setting an HDL breakpoint

bp <filename> <line_number> [-id <id_number>] [-disable]
[-cond <condition_expression>] [<command>…]

Querying a breakpoint

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

Arguments

- <filename>

  (required for an HDL breakpoint) A string that specifies the name of the source file in which to set the breakpoint.

- <line_number>

  (required for an HDL breakpoint) A string that specifies the line number at which the breakpoint is to be set.

- -id <id_number>

  (optional) A switch and argument pair that attempts to assign this id number to the breakpoint. The command returns an error if the id number you specify is already used.

Note

Ids for breakpoints are assigned from the same pool as those used for the when command. So, even if you haven't used an id number for a breakpoint, it's possible it is used for a when command.

- -disable

  (optional) A switch that sets the breakpoint to a disabled state. You can enable the breakpoint later using the enablebp command. This command enables breakpoints by default.

- -cond <condition_expression>

  (optional) A switch and argument pair that specifies condition(s) that determine whether the breakpoint is hit. You must enclose the condition expression within quotation marks (").

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

  NOTE: You can also specify this expression by choosing Tools > Breakpoints... from the main menu and using the Modify Breakpoints dialog box. Refer to Modifying File-Line Breakpoints in the User's Manual for more information.

The condition can be an expression with these operators:

	Operator
equals	==, =
not equal	!=, /=
AND	&&, AND
OR	||, OR

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

```
condition ::= Name | { expression }

expression ::= expression AND relation
               | expression OR relation
               | relation
relation ::= Name = Literal
           | Name /= Literal
           | Name ' EVENT
           | ( expression )
Literal ::= '<char>' | "<bitstring>" | <bitstring>
```

The "=" operator can occur only between a Name and a Literal. This means that you cannot compare the value of two signals; i.e., Name = Name is not possible.

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.

- <command>…

  (optional) A string, enclosed in braces ({}) that specifies one or more commands that are to be executed at the breakpoint. You must separate multiple commands by semicolons (;) or placed on multiple lines.

  NOTE: You can also specify this command string by choosing Tools > Breakpoints... from the main menu and using the Modify Breakpoints dialog box. Refer to Modifying File-Line Breakpoints in the User's Manual for more information.

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

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

- -query <filename> [<line_number> ...]

  (optional) A switch and argument group that returns information about the breakpoints set in the specified file. The information returned varies depending on which arguments you specify. The output contains six pieces of information, for example:

  ```
  bp -query top.vhd 70
  # 1 1 top.vhd 70 2 1
  ```

  - {1 | 0} — Indicates whether a breakpoint exists at the location.
  - 1 — always reports a 1
  - <file_name>
  - <line_number>
  - <id_number>
  - {1 | 0} — Indicates whether the breakpoint is enabled?

  If you specify this command with no arguments, it returns a list of all breakpoints in the design containing the following information, for example:

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

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

Examples

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

  ```
  bp
  ```

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

  ```
  bp alu.vhd 147
  ```

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

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

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

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

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

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

- Set a breakpoint so that the simulation pauses whenever clk=1 and prdy=0:

```
bp test.vhd 14 -cond "clk=1 AND prdy=0"
```

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

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

- List details about the breakpoint on line 48.

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

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

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

Note

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

See also

bd disablebp enablebp onbreak when

cd

The **cd** command changes the ModelSim local directory to the specified directory.

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

Syntax

cd [<dir>]

Arguments

- <dir>

  (optional) A string that specifies a full or relative directory path to which to change. If you do not specify a directory, the command changes to your home directory.

change

The **change** command modifies the value of a:

- VHDL constant, generic, or variable
- Verilog register or variable

Syntax

change <variable> <value>

Arguments

- <variable>

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

  You cannot use Wildcards.

  The following sections list supported objects:

  - VHDL
    - Scalar variable, constant, or generics of all types except FILE.

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

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

- <value>

  (required) A string that defines a value for the <variable>. The specified value must be appropriate for the type of the variable. You must enclose any <value> that contain spaces within quotation marks or curly braces.

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

Examples

- Change the value of the variable *count* to the hexadecimal value FFFF.

  ```
  change count 16#FFFF
  ```

- Change the value of the element of *rega* that is specified by the index (i.e., 16).

  ```
  change {rega[16]} 0
  ```

- Change the value of the set of elements of foo that is specified by the slice (i.e., 20:22).

  ```
  change {foo[20:22]} 011
  ```

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

  ```
  change file_name \"test2.txt\"
  ```

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

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

See also

force

configure

The **configure** command invokes the List or Wave widget configure command for the current default List or Wave window.

To change the default window, use the view command.

Syntax

configure list | wave [<option> <value>]

---- List Window Arguments
[-delta [all | collapse | none]] [-gateduration [<duration_open>]] [-gateexpr [<expression>]]
[-usegating [<value>]] [-strobeperiod [<period>]] [-strobestart [<start_time>]]
[-usesignaltriggers [<value>]] [-usestrobe [<value>]]

---- Wave Window Arguments
[-childrowmargin [<pixels>]] [-cursorlockcolor [<color>]] [-gridauto [off | on]]
[-gridcolor [<color>]][-griddelta [<pixels>]] [-gridoffset [<time>]] [-gridperiod [<time>]]
[-namecolwidth [<width>]] [-rowmargin [<pixels>]] [-signalnamewidth [<value>]]
[-timecolor [<color>]] [-timeline [<value>]]
[-timelineunits [fs | ps | ns | us | ms | sec | min | hr]] [-valuecolwidth [<width>]]
[-vectorcolor [<color>]] [-waveselectcolor [<color>]] [-waveselectenable [<value>]]

Description

The command works in three modes:

- without options or values it returns a list of all attributes and their current values
- with just an option argument (without a value) it returns the current value of that attribute
- with one or more option-value pairs it changes the values of the specified attributes to the new values

The returned information has five fields for each attribute: the command-line switch, the Tk widget resource name, the Tk class name, the default value, and the current value.

Arguments

- list | wave

  Specifies either the List or Wave widget to configure. Required.

- <option> <value>

  -bg <color> — Specifies the window background color. Optional.

  -fg <color> — Specifies the window foreground color. Optional.

  -selectbackground <color> — Specifies the window background color when selected. Optional.

-selectforeground <color> — Specifies the window foreground color when selected. Optional.

-font <font> — Specifies the font used in the widget. Optional.

-height <pixels> — Specifies the height in pixels of each row. Optional.

Arguments, List window only

- -delta [all | collapse | none]

  The **all** option displays a new line for each time step on which objects change; **collapse** displays the final value for each time step; and **none** turns off the display of the delta column. To use **-delta**, **-usesignaltriggers** must be set to 1 (on). Optional.

- -gateduration [<duration_open>]

  The duration for gating to remain open beyond when **-gateexpr** (below) becomes false, expressed in x number of timescale units. Extends gating beyond the back edge (the last list row in which the expression evaluates to true). Optional. The default value for normal synchronous gating is zero. If **-gateduration** is set to a non-zero value, a simulation value will be displayed after the gate expression becomes false (if you don't want the values displayed, set **-gateduration** to zero).

- -gateexpr [<expression>]

  Specifies the expression for trigger gating. Optional. (Use the **-usegating** argument to enable trigger gating.) The expression is evaluated when the List window would normally have displayed a row of data.

- -usegating [<value>]

  Enables triggers to be gated on (a value of 1) or off (a value of 0) by an overriding expression. Default is off. Optional. (Use the **-gatexpr** argument to specify the expression.) Refer to "Using Gating Expressions to Control Triggering" for additional information on using gating with triggers.

- -strobeperiod [<period>]

  Specifies the period of the list strobe. When using a time unit, the time value and unit must be placed in curly braces. Optional.

- -strobestart [<start_time>]

  Specifies the start time of the list strobe. When using a time unit, the time value and unit must be placed in curly braces. Optional.

- -usesignaltriggers [<value>]

  If 1, uses signals as triggers; if 0, not. Optional.

- -usestrobe [<value>]

  If 1, uses the strobe to trigger; if 0, not. Optional.

Arguments, Wave window only

- -childrowmargin [<pixels>]

  Specifies the distance in pixels between child signals. Optional. Default is 2. Related Tcl variable is PrefWave(childRowMargin).

- -cursorlockcolor [<color>]

  Specifies the color of a locked cursor. Default is red. Related Tcl variable is PrefWave(cursorLockColor).

- -gridauto [off | on]

  Controls the grid period when in simulation time mode.

  off — (default) user-specified grid period is used.

  on — grid period is determined by the major tick marks in the time line.

- -gridcolor [<color>]

  Specifies the background grid color; the default is grey50. Optional. Related Tcl variable is PrefWave(gridColor).

- -griddelta [<pixels>]

  Specifies the closest (in pixels) two grid lines can be drawn before intermediate lines will be removed. Optional. Default is 40. Related Tcl variable is PrefWave(gridDelta).

- -gridoffset [<time>]

  Specifies the time (in user time units) of the first grid line. Optional. Default is 0. Related Tcl variable is PrefWave(gridOffset).

- -gridperiod [<time>]

  Specifies the time (in user time units) between subsequent grid lines. Optional. Default is 1. Related Tcl variable is PrefWave(gridPeriod).

- -namecolwidth [<width>]

  Specifies in pixels the width of the name column. Optional. Default is 150. Related Tcl variable is PrefWave(nameColWidth).

- -rowmargin [<pixels>]

  Specifies the distance in pixels between top-level signals. Default is 4. Related Tcl variable is PrefWave(rowMargin).

- -signalnamewidth [<value>]

  Controls the number of hierarchical regions displayed as part of a signal name shown in the pathname pane. Optional. Default of 0 displays the full path. 1 displays only the leaf path element, 2 displays the last two path elements, and so on. Related Tcl variable is PrefWave(SignalNameWidth). Can also be set with the WaveSignalNameWidth variable in the *modelsim.ini* file.

- -timecolor [<color>]

  Specifies the time axis color. Default is green. Optional. Related Tcl variable is PrefWave(timeColor).

- -timeline [<value>]

  Specifies whether the horizontal axis displays simulation time (default) or grid period count. Default is zero. When set to 1, the grid period count is displayed. Related Tcl variable is PrefWave(timeline).

- -timelineunits [fs | ps | ns | us | ms | sec | min | hr]

  Specifies units for timeline display (does not affect the currently-defined simulation time). Default is ns.

- -valuecolwidth [<width>]

  Specifies in pixels the width of the value column. Default is 100. Related Tcl variable is PrefWave(valueColWidth).

- -vectorcolor [<color>]

  Specifies the vector waveform color. Default is #b3ffb3. Optional. Related Tcl variable is PrefWave(vectorColor).

- -waveselectcolor [<color>]

  Specifies the background highlight color of a selected waveform. Default is grey30. Related Tcl variable is PrefWave(waveSelectColor).

- -waveselectenable [<value>]

  Specifies whether the waveform background highlights when an object is selected. 1 enables highlighting; 0 disables highlighting. Default is 0. Related Tcl variable is PrefWave(waveSelectEnabled).

There are more options than are listed here. See the output of a configure list or configure wave command for all options.

Examples

- Display the current value of the 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
  ```

See also

view, Simulator GUI Preferences

<coverage_type> =

]

Important: When the **-metric aggregate** argument is used, the resulting metric number will not “match” any other total coverage number produced by other verification tools (i.e. coverage analyze). This is important because when you use any of the arguments (**-totals**, **-goal**, with ranktest command, the aggregate metric is the default.

dataset alias

The **dataset alias** command assigns an additional name (alias) to a dataset. The dataset can then be referenced by that alias. A dataset can have any number of aliases, but all dataset names and aliases must be unique.

Syntax

dataset alias <dataset_name> [<alias_name>]

Arguments

- <dataset_name>

  Specifies the name of the dataset to which to assign the alias. Required.

- <alias_name>

  Specifies the alias name to assign to the dataset. Optional. If you don't specify an alias_name, ModelSim lists current aliases for the specified dataset_name.

See also

dataset clear, dataset close, dataset config, dataset info, dataset list, dataset open, dataset rename, dataset restart, dataset save, dataset snapshot

dataset clear

The **dataset clear** command removes all event data from the current simulation WLF file while keeping all currently logged signals logged. Subsequent run commands will continue to accumulate data in the WLF file.

Syntax

dataset clear

Example

- 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
```

See also

dataset alias, dataset close, dataset config, dataset info, dataset list, dataset open, dataset rename, dataset restart, dataset save, dataset snapshot, "Recording Simulation Results With Datasets", log

dataset close

The **dataset close** command closes an active dataset. To open a dataset, use the **dataset open** command.

Syntax

dataset close <logicalname> | [-all]

Arguments

- <logicalname>

  Specifies the logical name of the dataset or alias you wish to close. Required if -all isn't used.

- -all

  Closes all open datasets including the simulation. Optional.

See also

dataset alias, dataset clear, dataset config, dataset info, dataset list, dataset open, dataset rename, dataset restart, dataset save, dataset snapshot

dataset config

The **dataset config** command configures WLF file parameters after a WLF file has already been opened.

Syntax

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

Arguments

- <dataset_name>

  Specifies the logical name of the dataset or alias you wish to configure. Required.

- -wlfcachesize <n>

  Sets the size in megabytes of the WLF reader cache. Optional. Does not affect the WLF write cache.

- -wlfdeleteonquit

  When set to 1 (enabled), deletes the WLF file automatically when the simulation exits. Optional. Valid for the current simulation dataset only.

- -wlfopt

  When set to 1 (enabled), optimizes the display of waveforms in the Wave window. Default. Optional.

See also

dataset alias, dataset clear, dataset close, dataset info, dataset list, dataset open, dataset rename, dataset restart, dataset save, dataset snapshot, "WLF File Parameter Overview", vsim

dataset info

The **dataset info** command reports a variety of information about a dataset.

Syntax

dataset info <option> <dataset_name>

Arguments

- <option>

  Identifies what information you want reported. Required. Only one option per command is allowed. The current options include:

  name — Returns the actual name of the dataset. Useful for identifying the real dataset name of an alias.

  file — Returns the name of the WLF file associated with the dataset.

  exists — Returns "1" if the dataset exists; "0" if it doesn't.

- <dataset_name>

  Specifies the name of the dataset or alias for which you want information. Optional. If you do not specify a dataset name, ModelSim uses the dataset of the current environment (see the environment command).

See also

dataset alias, dataset clear, dataset close, dataset config, dataset list, dataset open, dataset rename, dataset restart, dataset save, dataset snapshot

dataset list

The **dataset list** command lists all active datasets.

Syntax

dataset list [-long]

Arguments

- -long

  Lists the filename corresponding to each dataset's logical name. Optional.

See also

dataset alias, dataset clear, dataset close, dataset config, dataset info, dataset open, dataset rename, dataset restart, dataset save, dataset snapshot

dataset open

The **dataset open** command opens a WLF file (representing a prior simulation) and assigns it the logical name that you specify. To close a dataset, use **dataset close**.

Syntax

dataset open <filename> [<logicalname>]

Arguments

- <filename>

  Specifies the WLF file to open as a view-mode dataset. Required.

- <logicalname>

  Specifies the logical name for the dataset. Optional. This is a prefix that will identify the dataset in the current session. By default the dataset prefix will be the name of the specified WLF file.

Examples

- Open the dataset file *last.wlf* and assigns it the logical name *test*.

```
dataset open last.wlf test
```

See also

dataset alias, dataset clear, dataset close, dataset config, dataset info, dataset list, dataset rename, dataset restart, dataset save, dataset snapshot, vsim -view option

dataset rename

The **dataset rename** command changes the logical name of a dataset to the new name you specify.

Syntax

dataset rename <logicalname> <newlogicalname>

Arguments

- <logicalname>

  Specifies the existing logical name of the dataset. Required.

- <newlogicalname>

  Specifies the new logical name for the dataset. Required.

Examples

- Rename the dataset file "test" to "test2".

```
dataset rename test test2
```

See also

dataset alias, dataset clear, dataset close, dataset config, dataset info, dataset list, dataset open, dataset restart, dataset save, dataset snapshot

dataset restart

The **dataset restart** command unloads the specified dataset or current dataset and reloads the file using the same pathname. Wave window contents are restored after the reload.

Syntax

dataset restart [<filename>]

Arguments

- <filename>

  Specifies the WLF file to open as a view-mode dataset. Optional. If <filename> is not specified, the current dataset is restarted.

See also

dataset alias, dataset clear, dataset close, dataset config, dataset info, dataset list, dataset open, dataset rename, dataset save, dataset snapshot

dataset save

The **dataset save** command writes data from the current simulation to the specified file. This lets you save simulation data while the simulation is still in progress.

Syntax

dataset save <datasetname> <filename>

Arguments

- <datasetname>

  Specifies the name of the dataset you want to save. Required.

- <filename>

  Specifies the name of the file to save. Required.

Examples

- Save all current log data in the sim dataset to the file "gold.wlf".

```
dataset save sim gold.wlf
```

See also

dataset alias, dataset clear, dataset close, dataset config, dataset info, dataset list, dataset open, dataset rename, dataset restart, dataset snapshot

dataset snapshot

The **dataset snapshot** command saves data from the current WLF file (*vsim.wlf* by default) at a specified interval. This lets you take sequential or cumulative "snapshots" of your simulation data.

Syntax

dataset snapshot [-dir <directory>] [-disable] [-enable] [-file <filename>]
[-filemode overwrite | increment] [-mode cumulative | sequential] [-report] [-reset]
[-size <file size> | -time <simulation time>]

Arguments

- -dir <directory>

  Specifies a directory into which the files should be saved. Optional. Default is to save into the directory where ModelSim is writing the current WLF file.

- -disable

  Turns snapshotting off. Optional. All other options are ignored if you specify **-disable**.

- -enable

  Turns snapshotting on. Optional. Default.

- -file <filename>

  Specifies the name of the file to save. Optional. Default is "vsim_snapshot". ".wlf" will be appended to the file and possibly an incrementing suffix if **-filemode** is set to "increment".

- -filemode overwrite | increment

  Specifies whether to overwrite the snapshot file each time a snapshot occurs. Optional. Default is "overwrite". If you specify "increment", a new file is created for each snapshot. An incrementing suffix (1 to n) is added to each new file (e.g., *vsim_snapshot_1.wlf*).

- -mode cumulative | sequential

  Specifies whether to keep all data from the time signals are first logged. Optional. Default is "cumulative". If you specify "sequential", the current WLF file is cleared every time a snapshot is taken. See the examples for further details.

- -report

  Lists current snapshot settings in the Transcript pane. Optional. All other options are ignored if you specify **-report**.

- -reset

  Resets values back to defaults. Optional. The behavior is to reset to the default, then apply the remainder of the arguments on the command line. See examples below. If specified by itself without any other arguments, -reset disables dataset snapshot.

- -size <file size>

  Specifies that a snapshot occurs based on WLF file size. You must specify either **-size** or **-time**. See examples below.

- -time <simulation time>

  Specifies that a snapshot occurs based on simulation time. You must specify either **-time** or **-size**. See examples below.

Examples

- Create the file *vsim_snapshot.wlf* that is written to every time the current WLF file reaches a multiple of 10 MB (i.e., at 10 MB, 20 MB, 30 MB, etc.).

  ```
  dataset snapshot -size 10
  ```

- Similar to the previous example, but in this case the current WLF file is cleared every time it reaches 10 MB.

  ```
  dataset snapshot -size 10 -mode sequential
  ```

- Assuming simulator time units are ps, this command saves a file called *gold_n.wlf* every 1000000 ps. If you ran for 3000000 ps, you'd have three files: *gold_1.wlf* with data from 0 to 1000000 ps, *gold_2.wlf* with data from 1000001 to 2000000, and *gold_3.wlf* with data from 2000001 to 3000000.

  ```
  dataset snapshot -time 1000000 -file gold.wlf -mode sequential
  -filemode increment
  ```

Note

Because this example uses "sequential" mode, if you ran the simulation for 3500000 ps, the resulting *vsim.wlf* (the default log file) file will contain data only from 3000001 to 3500000 ps.

- Enable snapshotting with time=10000 and default mode (cumulative) and default filemode (overwrite).

  ```
  dataset snapshot -reset -time 10000
  ```

See also

dataset alias, dataset clear, dataset close, dataset config, dataset info, dataset list, dataset open, dataset rename, dataset restart, dataset save

delete

The **delete** command removes objects from either the List or Wave window.

Syntax

delete list | wave [-window <wname>] <object_name>

Arguments

- list | wave

  Specifies the target window for the **delete** command. Required.

- -window <wname>

  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). Optional. If no window is specified the default window is used; the default window is determined by the most recent invocation of the view command.

- <object_name>

  Specifies the name of an object. Required. Must match an object name used in an add list or add wave command. Multiple object names may be specified. Wildcard characters are allowed.

Examples

- Remove the object *vec2* from the list2 window.

```
delete list -window list2 vec2
```

See also

add list, add wave, and Wildcard Characters

describe

The **describe** command displays information about the specified HDL object or design region.

The description is displayed in the Transcript pane. The following kinds of objects can be described:

- Design region
- **VHDL** — signals, variables, and constants
- **Verilog** — nets and registers

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

Syntax

describe <name>

Arguments

- <name>

  The name of an HDL object for which you want a description. HDL object names can be full hierarchical names or relative names.

Examples

- Print the types of the three specified signals.

```
describe clk prw prdy
```

disablebp

The **disablebp** command turns off breakpoints and **when** commands. To turn the breakpoints or when statements back on again, use the **enablebp** command.

Syntax

disablebp [<id#>]

Arguments

- <id#>

  Specifies a breakpoint or **when** command id to disable. Optional. If you don't specify an id#, all breakpoints are disabled.

See also

bd, bp, enablebp, onbreak, resume, when

do

The **do** command executes commands contained in a macro file.

A macro file can have any name and extension. An error encountered during the execution of a macro file causes its execution to be interrupted, unless an onerror command, onbreak command, or the OnErrorDefaultAction Tcl variable has specified with the resume command.

Syntax

do <filename> [<parameter_value>]

Arguments

- <filename>

  Specifies the name of the macro file to be executed. Required. The name can be a pathname or a relative file name.

  Pathnames are relative to the current working directory if the **do** command is executed from the command line. If the **do** command is executed from another macro file, pathnames are relative to the directory of the calling macro file. This allows groups of macro files to be moved to another directory and still work.

- <parameter_value>

  Specifies values that are to be passed to the corresponding parameters $1 through $9 in the macro file. Optional. Multiple parameter values must be separated by spaces.

  If you want to make the parameters optional (i.e., specify fewer parameter values than the number of parameters actually used in the macro), you must use the argc simulator state variable in the macro. Refer to "Making Macro Parameters Optional".

  Note that there is no limit on the number of parameters that can be passed to macros, but only nine values are visible at one time. You can use the shift command to see the other parameters.

Examples

- This command executes the file *macros/stimulus*, passing the parameter value 100 to $1 in the macro file.

  ```
  do macros/stimulus 100
  ```

- If the macro file *testfile* contains the line **bp** $1 $2, this command would place a breakpoint in the source file named *design.vhd* at line 127.

  ```
  do  testfile  design.vhd  127
  ```

See also

"Tcl and Macros (DO Files)", "Modes of Operation", "Using a Startup File", DOPATH variable

drivers

The **drivers** command displays the names and strength of all drivers of the specified object.

The driver list is expressed relative to the top-most design signal/net connected to the specified object. If the object is a record or array, each subelement is displayed individually.

Syntax

drivers <object_name>

Arguments

- <object_name>

  Specifies the name of the signal or net whose drivers are to be shown. Required. All signal or net types are valid. Multiple names and wildcards are accepted.

Example

```
drivers /top/dut/pkt_cnt(4)
# Drivers for /top/dut/pkt_cnt(4):
#    St0  : Net /top/dut/pkt_cnt[4]
#      St0 : Driver /top/dut/pkt_counter/#IMPLICIT-WIRE(cnt_out)#6
```

In some cases, the output may supply a strength value similar to 630 or 52x, which indicates an ambiguous verilog strength. For more information, please refer to the Verilog LRM Std 1365-2005 section 7.10.2 "Ambiguous strengths: sources and combinations".

See also

readers

dumplog64

The **dumplog64** 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.

The **dumplog64** command cannot be used in a DO file.

Syntax

dumplog64 <filename>

Arguments

- <filename>

  The name of the WLF file to be read. Required.

echo

The **echo** command displays a specified message in the Transcript pane.

Syntax

echo [<text_string>]

Arguments

- <text_string>

  Specifies the message text to be displayed. Optional. If the text string is surrounded by quotes, blank spaces are displayed as entered. If quotes are omitted, two or more adjacent blank spaces are compressed into one space.

Examples

- If the current time is 1000 ns, this command:

  ```
  echo “The time is       $now ns.”
  ```

  produces the message:

  ```
  The time is       1000 ns.
  ```

- If the quotes are omitted:

  ```
  echo The time is       $now ns.
  ```

  all blank spaces of two or more are compressed into one space.

  ```
  The time is $now ns.”
  ```

- **echo** can also use command substitution, such as:

  ```
  echo The hex value of counter is [examine -hex counter].
  ```

  If the current value of counter is 21 (15 hex), this command produces:

  ```
  The hex value of counter is 15.
  ```

edit

The **edit** command invokes the editor specified by the EDITOR environment variable. By default, the specified filename will open in ModelSim Source editor.

Syntax

edit [<filename>]

Arguments

- <filename>

  Specifies the name of the file to edit. Optional. If the <filename> is omitted, the editor opens the current source file. If you specify a non-existent filename, it will open a new file.

See also

notepad, EDITOR environment variable

enablebp

The **enablebp** command turns on breakpoints and **when** commands that were previously disabled.

Syntax

enablebp [<id#>]

Arguments

- <id#>

  Specifies a breakpoint or when statement id to enable. Optional. If you don't specify an id#, all breakpoints are enabled.

See also

bd, bp, disablebp, onbreak, resume, when

environment

The **environment**, or **env** command, allows you to display or change the current dataset and region/signal environment.

Syntax

environment [-dataset] [-nodataset] [<pathname> | -forward | -back]

Arguments

- -dataset

  Displays the specified environment pathname *with* a dataset prefix. Optional. Dataset prefixes are displayed by default if more than one dataset is open during a simulation session.

- -nodataset

  Displays the specified environment pathname *without* a dataset prefix. Optional.

- <pathname>

  Specifies the pathname to which the current region/signal environment is to be changed. See Object Name Syntax for information on specifying pathnames. Optional.

  If omitted the command causes the pathname of the current region/signal environment to be displayed.

- -forward

  Displays the next environment in your history of visited environments. Optional.

- -back

  Displays the previous environment in your history of visited environments. Optional.

  Refer to the section "Setting your Context by Navigating Source Files" in the User's Manual for more information about -forward and -back.

Examples

- Display the pathname of the current region/signal environment.

  ```
  env
  ```

- Change all unlocked windows to the context of the "test" dataset.

  ```
  env-dataset 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 /
```

examine

The **examine** command examines one or more objects and displays current values (or the values at a specified previous time) in the Transcript pane.

The following objects can be examined:

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

To display a previous value, specify the desired time using the **-time** option.

Virtual signals and functions may also be examined within the GUI (actual signals are examined in the kernel).

The following rules are used by the examine command to locate an HDL object:

- If the name does not include a dataset name, then the current dataset is used.
- If the name does not start with a path separator, then the current context is used.
- If the name is a path separator followed by a name that is not the name of a top-level design unit, then the first top-level design unit in the design is used.
- For a relative name containing a hierarchical path, if the first object name cannot be found in the current context, then an upward search is done up to the top of the design hierarchy to look for a matching object name.
- If no objects of the specified name can be found in the specified context, then an upward search is done to look for a matching object in any visible enclosing scope up to an instance boundary. If at least one match is found within a given context, no (more) upward searching is done; therefore, some objects that may be visible from a given context will not be found when wildcards are used if they are within a higher enclosing scope.
- The wildcards '*' and '?' can be used at any level of a name except in the dataset name and inside of a slice specification.
- A wildcard character will never match a path separator. For example, */dut/** will match */dut/siga* and */dut/clk*. However, */dut** won't match either of those.

See Design Object Names for more information on specifying names.

Syntax

examine [-delta <delta>] [-env <path>] [-handle] [-in] [-out] [-inout] [-internal]
[-maxlen [0 | <integer>]] [-ports] [-expr <expression>] [-name]
[-<radix_type>] [-radix <type>] [-time <time>] [-value] <name>…

Arguments

- -delta <delta>

  Specifies a simulation cycle at the specified time from which to fetch the value. Optional. The default is to use the last delta of the time step. 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 delta. This option can be used only with objects that have been logged via the add list, add wave, or log command.

- -env <path>

  Specifies a path in which to look for an object name. Optional.

- -expr <expression>

  Specifies an expression to be evaluated. Optional. 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 evaluate the specified expression. If the **-time** argument is present, the expression will be evaluated at the specified time, otherwise it will be evaluated at the current simulation time. See GUI_expression_format for the format of the expression. The expression must be placed within curly braces.

- -handle

  Returns the memory address of the specified <name>. This value can be useful, as a semi-unique tag, for advanced HDL designers when analyzing the simulation of their design. This value is also used as the title of a box in the Watch window. This option will not return any value if you are in -view mode.

- -in

  Specifies that <name> include ports of mode IN. Optional.

- -out

  Specifies that <name> include ports of mode OUT. Optional.

- -inout

  Specifies that <name> include ports of mode INOUT. Optional.

- -internal

  Specifies that <name> include internal (non-port) signals. Optional.

- -maxlen [0 | <*integer*>]

  Specifies the maximum number of characters in the output of the command. The default setting is the value of the MaxValueLen simulator variable, which, itself, defaults to 30,000 characters. A value of zero (0) indicates an unlimited number of characters.

- -ports

  Specifies that <name> include all ports. Optional. Has the same effect as specifying -in, -inout, and -out together.

- -name

  Displays object name(s) along with the value(s). Optional. Default is **-value** behavior (see below).

- -<radix_type>

  Specifies the radix type for the objects that follow in the command. Valid entries (or any unique abbreviations) are: binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default. If no radix is specified for an enumerated type, the default representation is used. You can change the default radix for the current simulation using the radix command. You can change the default radix permanently by editing the DefaultRadix variable in the *modelsim.ini* file.

- -radix <type>

  Specifies a user-defined radix. Optional. Valid entries for <radix_type> are: binary, ascii, unsigned, decimal, octal, hex, symbolic, time, and default. The -radix <radix_type> argument can be used in place of the -<radix> entries. For example, -radix hexadecimal is the same as -hex.

- -time <time>

  Specifies the time value between 0 and $now for which to examine the objects. Optional. 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.

  If the <time> field uses a unit, the value and unit must be placed in curly braces. For example, the following are equivalent for ps resolution:

  ```
  exa -time {3.6 ns} signal_a
  exa -time 3600 signal_a
  ```

- -value

  Returns value(s) as a curly-braces separated Tcl list. Default. Use to toggle off a previous use of **-name**.

- <name>…

  Specifies the name of any HDL object. Required. All object types are allowed, except those of the type file. Multiple names and wildcards are accepted. Spaces, square brackets, and extended identifiers require curly braces; see examples below for more details. To examine a VHDL variable you can add a process label to the name. For example, (make certain to use two underscore characters):

  ```
  exa line__36/i
  ```

Examples

- Return the value of */top/bus1*.

  ```
  examine /top/bus1
  ```

- Return the value of the subelement of *rega* that is specified by the index (i.e., 16). Note that you must use curly braces when examining subelements.examine

```
{rega[16]}
```

- Return the value of the contiguous subelements of *foo* specified by the slice (i.e., 20:22). Note the curly braces.

```
examine {foo[20:22]}
```

- Note that when specifying an object that contains an extended identifier as the last part of the name, there must be a space after the closing '\' and before the closing '}'.

```
examine {/top/\My extended id\ }
```

- In this example, the **-expr** option specifies a signal path and user-defined Tcl variable. The expression will be evaluated at 3450us.

```
examine -time {3450 us} -expr {/top/bus and $bit_mask}
```

- Using the ${fifo} syntax limits the variable to the simple name fifo, instead of interpreting the parenthesis as part of the variable. Quotes are needed when spaces are involved; and by using quotes (") instead of braces, the Tcl interpreter will expand variables before calling the command.

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

- Because **-time** is not specified, this expression will be evaluated at the current simulation time. Note the signal attribute and array constant specified in the expression.

```
examine -expr {clk'event && (/top/xyz == 16'hffae)}
```

Commands like find and **examine** return their results as a Tcl list (just a blank-separated list of strings). You can do things like:

```
foreach sig [find sig ABC*] {echo "Signal $sig is [exa $sig]" …}

if {[examine -bin signal_12] == "11101111XXXZ"} {…}

examine -hex [find *]
```

See also

Design Object Names, Wildcard Characters

exit

The **exit** command exits the simulator and the ModelSim application.

If you want to stop the simulation using a when command, you must use a stop command within your when statement. DO NOT use an **exit** command or a quit command. The **stop** command acts like a breakpoint at the time it is evaluated.

Syntax

exit [-force] [-code]

Argument

- -force

  Quits without asking for confirmation. Optional; if this argument is omitted, ModelSim asks you for confirmation before exiting.

- -code <integer>

  Quits the simulation and issues an exit code.

  <integer> — This is the value of the exit code. You should not specify an exit code that already exists in the tool. Refer to the section "Exit Codes" in the User's Manual for a list of existing exit codes. You can also specify a variable in place of the <integer>.

  You should always print a message before executing the exit -code command to explicitly state the reason for exiting.

Examples

You can use the exit -code syntax to instruct a vmake run to exit when encountering an assertion error. The onbreak command can specify commands to be executed upon an assert failure of sufficient severity, upon which the simulator can be made to return an exit status, as 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

The **find** command locates objects in the design whose names match the name specification you provide. You must specify the type of object you want to find.

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. When searching for instances, the find command returns the primary design unit name.

When searching for nets and signals, the order in which arguments are specified is unimportant. When searching for virtuals, however, all optional arguments must be specified before any object names.

The following rules are used by the find command to locate an object:

- If the name does not include a dataset name, then the current dataset is used.
- If the name does not start with a path separator, then the current context is used.
- If the name is a path separator followed by a name that is not the name of a top-level design unit, then the first top-level design unit in the design is used.
- For a relative name containing a hierarchical path, if the first object name cannot be found in the current context, then an upward search is done up to the top of the design hierarchy to look for a matching object name.
- If no objects of the specified name can be found in the specified context, then an upward search is done to look for a matching object in any visible enclosing scope up to an instance boundary. If at least one match is found within a given context, no (more) upward searching is done; therefore, some objects that may be visible from a given context will not be found when wildcards are used if they are within a higher enclosing scope.
- The wildcards '*' and '?' can be used at any level of a name except in the dataset name and inside of a slice specification. Square bracket ’[]’ wildcards can also be used.
- A wildcard character will never match a path separator. For example, */dut/** will match */dut/siga* and */dut/clk*. However, */dut** won’t match either of those.
- Because square brackets are wildcards in the find command, only parentheses ’()’ can be used to index or slice arrays.
- The *WildcardFilter* Tcl preference variable is used by the find command to exclude the specified types of objects when performing the search.

See Design Object Names for more information on specifying names.

Syntax

find nets | signals [-in] [-inout] [-internal] <object_name> … [-nofilter] [-out] [-ports] [-recursive]

find instances | blocks [-recursive] <object_name> …

find virtuals [-kind <kind>] [-unsaved] <object_name> …

find classes [<class_name>]

find objects [-class <class_name>] [-isa <class_name>] [<object_name>]

Arguments for nets and signals

- -in

  Specifies that the scope of the search is to include ports of mode IN. Optional.

- -inout

  Specifies that the scope of the search is to include ports of mode INOUT. Optional.

- -internal

  Specifies that the scope of the search is to include internal (non-port) objects. Optional.

- <object_name> …

  Specifies the net or signal for which you want to search. Required. Multiple nets and signals and wildcard characters are allowed. Wildcards cannot be used inside of a slice specification. Spaces, square brackets, and extended identifiers require special syntax; see the examples below for more details.

- -nofilter

  Specifies that the *WildcardFilter* Tcl preference variable be ignored when finding signals or nets. Optional.

- -out

  Specifies that the scope of the search is to include ports of mode OUT. Optional.

- -ports

  Specifies that the scope of the search is to include all ports. Optional. Has the same effect as specifying **-in**, **-out**, and **-inout** together.

- -recursive

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

Arguments for instances and blocks

- -recursive

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

- <object_name> …

  Specifies the instance or block for which you want to search. Required. Multiple instances and wildcard characters are allowed.

Arguments for virtuals

- -kind <kind>

  Specifies the kind of virtual object for which you want to search. Optional. <kind> can be one of designs, explicits, functions, implicits, or signals.

- -unsaved

  Specifies that ModelSim find only virtuals that have not been saved to a format file.

- <object_name> …

  Specifies the virtual object for which you want to search. Required. Multiple virtuals and wildcard characters are allowed.

Arguments for classes

- <class_name>

  Specifies the incrTcl class for which you want to search. Optional. Wildcard characters are allowed. The options for class_name include nets, objects, signals, and virtuals. If you do not specify a class name, the command returns all classes in the current namespace context. See "incrTcl commands" in the Tcl Man Pages for more information.

Arguments for objects

- -class <class_name>

  Restricts the search to objects whose most-specific class is **class_name**. Optional.

- -isa <class_name>

  Restricts the search to those objects that have **class_name** anywhere in their heritage. Optional.

- <object_name>

  Specifies the incrTcl object for which you want to search. Optional. Wildcard characters are allowed. If you do not specify an object name, the command returns all objects in the current namespace context. See "incrTcl commands" in the Tcl Man Pages for more information.

Examples

- Find all signals in the entire design.

  ```
  find signals -r /*
  ```

- Find all input signals in region /top that begin with the letters "xy".

  ```
  find nets -in /top/xy*
  ```

- Find all signals in the design hierarchy at or below the region *<current_context>/u1/u2* whose names begin with "cl".

  ```
  find signals -r u1/u2/cl*
  ```

- Find a signal named *s1*. Note that you must enclose the object in curly braces because of the square bracket wildcard characters.

```
find signals {s[1]}
```

- Find signals *s1*, *s2*, or *s3*.

```
find signals {s[123]}
```

- Find the element of signal *s* that is indexed by the value 1. Note that the **find** command uses parentheses, not square brackets, to specify a subelement index.

```
find signals s(1)
```

- Find a 4-bit array named *data*. Note that you must use curly braces due to the spaces in the array slice specification.

```
find signals {/top/data(3 downto 0)}
```

- Note that when specifying an object that contains an extended identifier as the last part of the name, there must be a space after the closing '\' and before the closing '}'.

```
find signals {/top/\My extended id\ }
```

- If */dut/core/pclk* exists, prints the message "pclk does exist" in the transcript. This would typically be run in a Tcl script.

```
if {[find signals /dut/core/pclk] != ""} {
  echo "pclk does exist"
}
```

- Find instances based on their names using wildcards. Send search results to a text file that lists instance names, including the hierarchy path, on separate lines.

```
# Search for all instances with 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 ;
```

See also

Design Object Names, Wildcard Characters

find infiles

The **find infiles** command searches the specified files and prints to the Transcript pane those lines from the files that match the specified pattern.

You can double-click on the results in the Transcript pane to open the specific file and display the referenced line.

When you use this command in command-line mode, outside of the GUI, the results are sent to **stdout** and you do not have the capability to view the file by double-clicking the result.

Syntax

find infiles <string_pattern> {<file_pattern> [<file_pattern> ...]}

Arguments

- <string_pattern>

  The string you are searching for. You can use regular expression wildcards to further restrict the search capability.

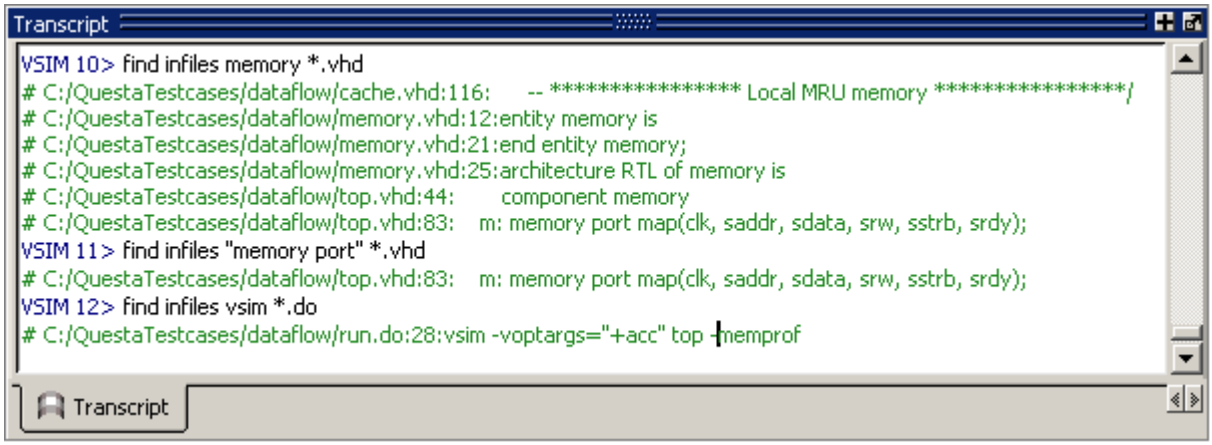
- <file_pattern> [<file_pattern> ...]

  The file(s) you are searching. You can use regular expression wildcards to further restrict the search capability.

Example

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

Figure 2-1. find infiles Example



find insource

The **find insource** command searches all source files related to the current design and prints to the Transcript pane those lines from the files that match the specified pattern.

You can double-click on the results in the Transcript pane to open the specific file and display the referenced line.

When you use this command in command-line mode, outside of the GUI, the results are sent to **stdout** and you do not have the capability to view the file by double-clicking the result.

Syntax

find insource <pattern>

Arguments

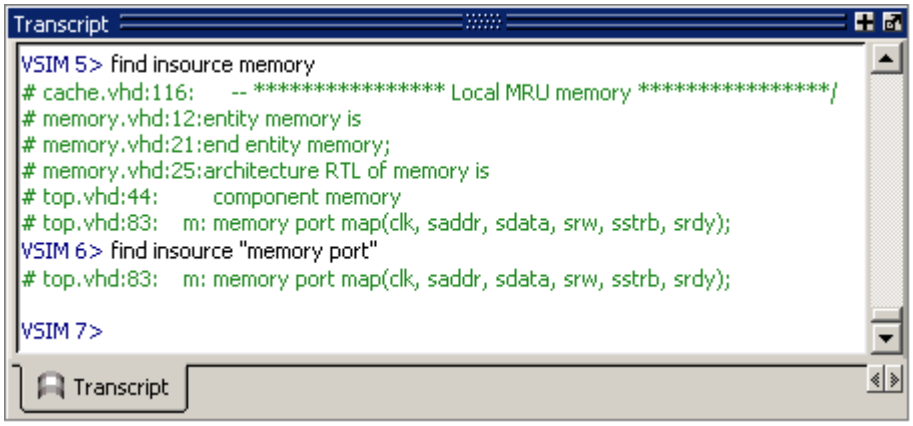
- <pattern>

  The string you are searching for. You can use regular expression wildcards to further restrict the search. You must enclose <pattern> in quotes (") if it includes spaces.

Example

Figure 2-2 shows a couple of examples of the find insource command and its results in the Transcript window.

Figure 2-2. find insource Example



formatTime

The formatTime command provides global format control for all time values displayed in the GUI. This command always returns the current state of its three arguments.

Syntax

formatTime +|-commas | +|-nodefunits | +|-bestunits

Arguments

- +|-commas

  Insert commas into the time value to make it easier to read. Optional. A leading '+' turns the argument on; a leading '-' turns the argument off. Default is off.

- +|-nodefunits

  Do not include default unit in the time. Optional. A leading '+' turns the argument on; a leading '-' turns the argument off. Default is off.

- +|-bestunits

  Use the largest unit value possible. Optional. A leading '+' turns the argument on; a leading '-' turns the argument off. Default is off.

Examples

- Display commas in time values.

  ```
  formatTime +commas
  ```

  Instead of displaying 6458131 ps, the GUI will display 6,458,131 ps.

- Use largest unit value possible.

  ```
  formatTime +bestunits
  ```

  Displays 8 us instead of 8,000 ns.

force

The **force** command allows you to apply stimulus interactively to VHDL signals and Verilog nets.

Since **force** commands (like all commands) can be included in a macro file, it is possible to create complex sequences of stimuli.

When you do not specify any arguments, this command returns a list of the most recently applied force commands.

There are a number of constraints on what you can and cannot force:

- You cannot force VHDL or Verilog variables (time or realtime); these must be changed. See the change command.
- In VHDL and mixed models, you cannot force an input port that is mapped at a higher level. In other words, you can force the signal at the top of the hierarchy connected to the input port but you cannot force the input port directly.
- You cannot force bits or slices of a register; you can force only the entire register.
- You cannot force a VHDL alias of a VHDL signal.
- You cannot force an input port that has a conversion function on the input.
- You can force "Virtual Signals" if the number of bits corresponds to the signal value. You cannot force virtual functions.

Syntax

force [-freeze | -drive | -deposit] [-cancel <time>] [-repeat <time>] <object_name> <value> [<time>] [, <value> <time> …]

Arguments

- -freeze

  Freezes the object at the specified value until it is forced again or until it is unforced with a noforce command. Optional.

- -drive

  Attaches a driver to the object and drives the specified value until the object is forced again or until it is unforced with a noforce command. Optional.

  This option is illegal for unresolved signals signals.

- -deposit

  Sets the object to the specified value. The value remains until there is a subsequent driver transaction, or until the object is forced again, or until it is unforced with a noforce command. Optional.

  If one of the **-freeze**, **-drive**, or **-deposit** options is 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 default force kind in the DefaultForceKind preference variable.

- -cancel <time>

  Cancels the **force** command at the specified **<time>**. The time is relative to the current time unless an absolute time is specified by preceding the value with the character @. Cancellation occurs at the last simulation delta cycle of a time unit. A value of zero cancels the force at the end of the current time period. Optional.

  ```
  -cancel 520 ns     \\ Relative Time
  -cancel @520 ns    \\ Absolute Time
  ```

- -repeat <time>

  Repeats the **force** command, where **<time>** is the time at which to start repeating the cycle. The time is relative to the current time. A repeating **force** command will force a value before other non-repeating **force** commands that occur in the same time step. Optional.

- <object_name>

  Specifies the name of the HDL object to be forced. Required. A wildcard is permitted only if it matches one object. See Design Object Names for the full syntax of an object name. The object name must specify a scalar type or a one-dimensional array of character enumeration. You may also specify a record subelement, an indexed array, or a sliced array, as long as the type is one of the above. Required.

- <value>

  Specifies the value to which the object is to be forced. The specified value must be appropriate for the type. Required.

  A VHDL one-dimensional array of character enumeration can be forced as a sequence of character literals or as a based number with a radix of 2, 8, 10 or 16. For example, the following values are equivalent for a signal of type bit_vector (0 to 3):

Value	Description
1111	character literal sequence
2#1111	binary radix
10#15	decimal radix
16#F	hexadecimal radix

Note

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

- <time>

  Specifies the time to which the value is to be applied. The time is relative to the current time unless an absolute time is specified by preceding the value with the character @. If the time units are not specified, then the default is the resolution units selected at simulation start-up. Optional.

  A zero-delay force command causes the change to occur in the current (rather than the next) simulation delta cycle.

Examples

- Force *input1* to 0 at the current simulator time.

  ```
  force input1 0
  ```

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

  ```
  force bus1 01XZ 100 ns
  ```

- Force *bus1* to 16#F at the absolute time 200 measured in the resolution units selected at simulation start-up.

  ```
  force bus1 16#f @200
  ```

- Force *input1* to 1 at 10 time units after the current simulation time and to 0 at 20 time units after the current simulation time. This cycle repeats starting at 100 time units after the current simulation time, so the next transition is to 1 at 100 time units after the current simulation time.

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

- Similar to the previous example, but also specifies the time units. Time unit expressions preceding the "-r" must be placed in curly braces.

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

- Force signal *s* to alternate between values 1 and 0 every 100 time units until time 1000. Cancellation occurs at the last simulation delta cycle of a time unit.

  ```
  force s 1 0, 0 100 -repeat 200 -cancel 1000
  ```

  So,

  ```
  force s 1 0 -cancel 0
  ```

  will force signal *s* to 1 for the duration of the current time period.

- Force *siga* to decimal value 85 whenever the value on the signal is 1.

  ```
  when {/mydut/siga = 10#1} {
    force -deposit /mydut/siga 10#85
  }
  ```

See also

noforce, change

Note

You can configure defaults for the force command by setting the **DefaultForceKind** variable in the *modelsim.ini* file. Refer to "Force Command Defaults".

help

The **help** command displays in the Transcript pane a brief description and syntax for the specified command.

Syntax

help [<command> | <topic>]

Arguments

- <command>

  Specifies the command for which you want help. The entry is case and space sensitive. Optional.

- <topic>

  Specifies a topic for which you want help. The entry is case and space sensitive. Optional. 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

The **history** command lists the commands you have executed during the current session. History is a Tcl command. For more information, consult the Tcl Man Pages.

Syntax

history [clear] [keep <value>]

Arguments

- clear

  Clears the history buffer. Optional.

- keep <value>

  Specifies the number of executed commands to keep in the history buffer. Optional. The default is 50.

layout

The **layout** command loads, saves, lists, or deletes custom GUI layouts.

The command options include:

- **layout load** opens the specified layout
- **layout save** saves the current layout to the specified name
- **layout names** lists all known layouts
- **layout current** lists the current layout
- **layout delete** removes the current layout from the *.modelsim* file (UNIX/Linux) or Registry (Windows)

See "Layouts and Modes of Operation" for more information.

Syntax

layout load <name>

layout save <name>

layout names

layout current

layout delete <name>

Arguments

- <name>

  Specifies the name of the layout. Required.

log

The **log** 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. The log is stored in a WLF file in the working directory. By default the file is named *vsim.wlf*. You can change the default name using the -wlf option of the vsim command.

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 is the source of data for the List and Wave windows. An object that has been logged and is subsequently added to the List or Wave window will have its complete history back to the start of logging available for listing and waving.

Syntax

log [-depth <level>] [-flush] [-howmany] [-in] [-inout] [-internal] [-out] [-ports] [-recursive] <object_name> …

Arguments

- -depth <level>

  Restricts a recursive search (specified with the **-recursive** argument) to a certain level of hierarchy. <level> is an integer greater than or equal to zero. For example, if you specify -depth 1, the command descends only one level in the hierarchy. Optional.

- -flush

  Adds region data to the WLF file after each individual log command. Optional. Default is to add region data to the log file only when a command that advances simulation time is executed (e.g., run, step, etc.) or when you quit the simulation.

- -howmany

  Returns an integer indicating the number of signals found. Optional.

- -in

  Specifies that the WLF file is to include data for ports of mode IN whose names match the specification. Optional.

- -inout

  Specifies that the WLF file is to include data for ports of mode INOUT whose names match the specification. Optional.

- -internal

  Specifies that the WLF file is to include data for internal (non-port) objects whose names match the specification. Optional.

- -out

  Specifies that the WLF file is to include data for ports of mode OUT whose names match the specification. Optional.

- -ports

  Specifies that the scope of the search is to include all ports. Optional.

- -recursive

  Specifies that the scope of the search is to descend recursively into subregions. Optional. If omitted, the search is limited to the selected region. You can use the **-depth** argument to specify how far down the hierarchy to descend.

- <object_name>

  Specifies the object name which you want to log. Required. Multiple object names may be specified. Wildcard characters are allowed. (Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.)

Examples

- Log all objects in the design.

  ```
  log -r /*
  ```

- Log all output ports in the current design unit.

  ```
  log -out *
  ```

See also

add list, add wave, nolog, "Recording Simulation Results With Datasets", and Wildcard Characters

Note

The log command is also known as the "add log" command.

lshift

The **lshift** command takes a Tcl list as an argument and shifts it in-place, one place to the left, eliminating the left-most element.

The number of shift places may also be specified. Returns nothing.

Syntax

lshift <list> [<amount>]

Arguments

- <list>

  Specifies the Tcl list to target with **lshift**. Required.

- <amount>

  Specifies the number of places to shift. Optional. Default is 1.

Examples

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

See also

See the Tcl man pages (**Help > Tcl Man Pages**) for details.

lsublist

The **lsublist** command returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern.

Syntax

lsublist <list> <pattern>

Arguments

- <list>

  Specifies the Tcl list to target with **lsublist**. Required.

- <pattern>

  Specifies the pattern to match within the <list> using Tcl glob-style matching. Required.

Examples

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

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

See also

The **set** command is a Tcl command. See the Tcl man pages (**Help > Tcl Man Pages**) for details.

mem compare

The **mem compare** command compare selected memory to reference memory or file. Must have the "diff" utility installed and visible in your search path in order to run the mem compare command.

Syntax

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

Arguments

- -mem <ref_mem>

  Specifies a reference memory to be compared.

- -file <ref_file>

  Specifies a reference file to be compared.

- actual_mem

  Specifies the name of the memory to be compared against the reference data.

mem display

The **mem display** command prints to the Transcript pane the memory contents of the specified instance. As a shorthand, if the given instance path contains only a single array signal or variable, the signal or variable name need not be specified.

Address radix, data radix, and address range for the output can also be specified, as well as special output formats.

You can redirect the output of the **mem display** command into a file for later use with the **mem load** command. The output file can also be read by the Verilog $readmem system tasks if the memory module is a Verilog module and Verilog memory format (hex or binary) is specified. 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.

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.

Syntax

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

Arguments

- -format [bin | hex | mti]

  Specifies the output format of the contents. Optional. The default format is mti. For details on mti format, see the description contained in mem load.

- -addressradix <radix_type>

  Specifies the address radix for the default (mti) formatted files. The <radix_type> can be specified as: d (decimal) or h (hex). Optional. If the output format is mti, the default is d.

- -dataradix <radix_type>

  Specifies the data radix for the default (mti) formatted files. Optional. If unspecified, the global default radix is used. Valid entries (or any unique abbreviations) are: binary, decimal, unsigned, octal, hex, symbolic, and default. If no radix is specified for an enumerated type, the symbolic representation is used. You can change the default radix type for the current simulation using the radix command. You can change the default radix permanently by editing the DefaultRadix variable in the modelsim.ini file.

- -wordsperline <Nwords>

  Specifies how many words are to be printed on each line, with the default assuming an 80 column display width. <Nwords> is an unsigned integer. Optional.

- -startaddress <st>

  Specifies the start address for a range of addresses to be displayed. The <st> can be specified as any valid address in the memory. Optional. If unspecified, the default is the start of the memory.

- -endaddress <end>

  Specifies the end address for a range of addresses to be displayed. The <end> can be specified as any valid address in the memory. Optional. If unspecified, the default is the end of the memory.

- -noaddress

  Specifies that addresses not be printed. Optional.

- -compress

  Specifies that identical lines not be printed. Optional. 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.

- <path>

  Specifies the full path to the memory instance. Optional. The default is the current context, as shown in the Structure tab of the Workspace. Indexes can be specified.

Examples

- This command displays the memory contents of instance */top/m/mru_mem*, addresses 5 to 10 to the screen as follows:

```
mem display -startaddress 5 -endaddress 10/top/c/mru_mem

# 5: 110 110 110 110 110 000
```

- Display the memory contents of the same instance to the screen in hex format, as follows:

```
mem display -format hex -startaddress 5 -endaddress 10
/top/c/mru_mem

# 5: 6 6 6 6 6 0
```

See Also

mem load

mem list

The **mem list** command displays a flattened list of all memory instances in the current or specified context after a design has been elaborated.

Each instance line is prefixed by "VHDL:" or "Verilog:", depending on the type of model.

Returns the signal/variable name, address range, and depth and width of the memory.

Syntax

mem list [-r] [<path>]

Arguments

- -r

  Recursively descends into sub-modules when listing memories. Optional.

- <path>

  The hierarchical path to the location the search should start. Optional. The default is the current context, as shown in the Structure tab of the Workspace pane.

Examples

- Recursively lists all memories at the top level of the design.

  ```
  mem list -r /
  ```

  Returns:

  ```
  # Verilog: /top/m/mem[0:255](256d x 16w)
  #
  ```

- Recursively lists all memories in */top2/uut*.

  ```
  mem list /top2/uut -r
  ```

  Returns:

  ```
  # Verilog: /top2/uut/mem[0:255] x 16w
  ```

mem load

The **mem load** command updates the simulation memory contents of a specified instance. You can upload contents either from a memory data file, a memory pattern, or both. If both are specified, the pattern is applied only to memory locations not contained in the file.

A relocatable memory file is one that has been saved without address information. You can load a relocatable memory file into the instance of a memory core by specifying an address range on the **mem load** command line. If no address range (starting and ending address) is specified, the memory is loaded starting at the first location.

The order in which the data is placed into the memory depends on the format specified by the **-format** option. If you choose bin or hex format, the memory is filled low to high, to be compatible with $readmem commands. This is in contrast to the default **mti** format, which fills the memory according to the memory declaration, from left index to right index.

For Verilog objects and VHDL integers and std_logic types: if the word width in a file is wider than the word width of the memory, the leftmost bits (msb's) in the data words are ignored. To allow wide words use the **-truncate** option which will ignore the msb bits that exceed the memory word size. If the word width in the file is less than the width of the memory, and the left-most digit of the file data is not 'X', then the left-most bits are zero filled. Otherwise, they are X-filled.

The type of data required for the **-filldata** argument is dependent on the **-filltype** specified: a fixed value, or one that governs an incrementing, decrementing, or random sequence.

- For fixed pattern values, the fill pattern is repeatedly tiled to initialize the memory block specified. The pattern can contain multiple word values for this option.
- For incrementing or decrementing patterns, each memory word is treated as an unsigned quantity, and each successive memory location is filled in with a value one higher or lower than the previous value. The initial value must be specified.
- For a random pattern, a random data sequence will be generated to fill in the memory values. The data type in the sequence will match the type stored in the memory. For std_logic and associated types, unsigned integer sequences are generated. A seed value may be specified on the command line. For any given seed, the generated sequence is identical.

The interpretation of the pattern data is performed according to the default system radix setting. However, this can be overridden with a standard Verilog-style '<radix_char><data> specification.

Syntax

mem load [-infile <infile> -format [bin | hex | mti]] [-filltype <filltype>]
[-filldata <patterndata>] [-fillradix <radix_type>] [-skip <Nwords>] [-truncate]
[-startaddress <st>] [-endaddress <end>] [<path>]

Arguments

- -infile <infile>

  Updates memory data from the specified file. Required unless the **-filltype** argument is used.

- -endaddress <end>

  Specifies the end address for a range of addresses to be loaded. The <end> can be specified as any valid address in the memory. Optional.

- -format [bin | hex | mti]

  Specifies the format of the file to be loaded. The <formtype> can be specified as: bin, hex, or mti. bin and hex are the standard Verilog hex and binary memory pattern file formats. These can be used with Verilog memories, and with VHDL memories composed of std_logic types.

  In the MTI memory data file format, internal file address and data radix settings are stored within the file itself. Thus, there is no need to specify these settings on the **mem load** command line. If a format specified on the command line and the format signature stored internally within the file do not agree, the file cannot be loaded.

- -filltype <filltype>

  Fills in memory data patterns algorithmically. The <filltype> can be specified as: value, inc, dec, or rand. Required unless the **-infile** argument is used, in which case it is optional. Default is value.

- -filldata <patterndata>

  Specifies the pattern parameters, value for fixed-value fill operations, and seed or starting point for random, increment, or decrement fill operations. Required if **-filltype** is used.

  A fill pattern covers any of the selected address range that is not populated from file values. If a fill pattern is used without a file option, the entire memory or specified address range is initialized with the fill pattern.

- -fillradix <radix_type>

  Specifies radix of the data specified by "-filldata" option. Valid entries (or any unique abbreviations) are: binary, decimal, unsigned, octal, hex, symbolic, and default.

- -skip <Nwords>

  Specifies the number of words to be skipped between each fill pattern value. <Nwords> is specified as an unsigned integer. Optional. Used with **-filltype** and **-filldata**.

- -truncate

  Ignores any most significant bits (msb) in a memory word which exceed the memory word size. By default, when memory word size is exceeded, an error results. Optional.

- -startaddress <st>

  Specifies the start address for a range of addresses to be loaded. The <st> can be specified as any valid address in the memory. Optional.

- <path>

  The hierarchical path to the memory instance. If the memory instance name is unique, shorthand instance names can be used. Optional. The default is the current context, as shown in the Structure tab of the Workspace pane.

  Memory address indexes can be specified in the instance name also. If addresses are specified both in the instance name and the file, only the intersection of the two address ranges is populated with memory data.

Examples

- Load the memory pattern from the file *vals.mem* to the memory instance */top/m/mem*, filling the rest of the memory with the fixed-value 1‘b0.

  ```
  mem load -infile vals.mem -format bin -filltype value -filldata 1‘b0
  /top/m/mem
  ```

  When you enter the **mem display** command on memory addresses 0 through 12, you see the following:

```
mem display -startaddress 0 -endaddress 12 /top/m/mem
#  0: 0000000000000000 0000000000000001 0000000000000010 0000000000000011
#  4: 0000000000000100 0000000000000101 0000000000000110 0000000000000111
#  8: 0000000000001000 0000000000001001 0000000000000000 0000000000000000
# 12: 0000000000000000
```

- Load the memory pattern from the file *vals.mem* to the memory instance */top/m/mru_mem*, filling the rest of the memory with the fixed-value 16’Hbeef.

  ```
  mem load -infile vals.mem -format hex -st 0 -end 12 -filltype value
  -filldata 16’Hbeef /top/m/mru_mem
  ```

- Load memory instance */top/mem2* with two words of memory data using the Verilog Hex format, skipping 3 words after each fill pattern sequence.

  ```
  mem load -filltype value -filldata "16’hab 16’hcd" /top/mem2 -skip 3
  ```

- Truncate the msb bits that exceed the maximum word size (specified in HDL code).

  ```
  mem load -format h -truncate -infile data_files/data.out
  /top/m_reg_inc/mem
  ```

See also

mem save

mem save

The **mem save** command saves the contents of a memory instance to a file in any of the supported formats: Verilog binary, Verilog hex, and MTI memory pattern data.

This command works identically to the **mem display** command, except that its output is written to a file rather than a display.

The order in which the data is placed into the saved file depends on the format specified by the **-format** argument. If you choose **bin** or **hex** format, the file is populated from low to high, to be compatible with $readmem commands. This is in contrast to the default **mti** format, which populates the file according to the memory declaration, from left index to right index.

You can use the **mem save** command to generate relocatable memory data files. The **-noaddress** option omits the address information from the memory data file. You can later load the generated memory data file using the **memory load** command.

Syntax

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

Arguments

- -format bin | hex | mti

  Specifies the output format. The <format_spec> can be specified as bin, hex, or mti. Optional. The default format is mti. The MTI memory pattern data format is described in mem load.

- -addressradix <radix_type>

  Specifies the address radix for the default mti formatted files. Optional. The <radix_type> can be specified as: dec or hex. The default is the decimal representation.

- -dataradix <radix_type>

  Specifies the data radix for the default mti formatted files. Optional. The <radix_type> can be specified as symbolic, binary, octal, decimal, unsigned, or hex. 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.

- -wordsperline <Nwords>

  Specifies how many memory values are to be printed on each line. Optional. The default assumes an 80 character display width. The <Nwords> is specified as an unsigned integer.

- -startaddress <st>

  Specifies the start address for a range of addresses to be saved. The <st> can be specified as any valid address in the memory. Optional.

- -endaddress <end>

  Specifies the end address for a range of addresses to be saved. The <end> can be specified as any valid address in the memory. Optional.

- -noaddress

  Prevents addresses from being printed. Optional. Mutually exclusive with the **-compress** option.

- -compress

  Specifies that only unique lines are printed, identical lines are not printed. Optional. Mutually exclusive with the **-noaddress** option.

- -outfile <filename>

  Specifies that the memory contents be stored in <filename>. Required.

- <path>

  The hierarchical path to the location of the memory instance. Optional. The default is the current context, as shown in the Structure tab of the Workspace pane.

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: 0000000000000100 0000000000000101 0000000000000110
  0000000000000111
  8: 0000000000001000 0000000000001001 xxxxxxxxxxxxxxxx
  ```

See also

mem display, mem load

mem search

The **mem search** command finds and prints to the screen the first occurring match of a specified memory pattern in the specified memory instance. Shorthand instance names are accepted.

Optionally, you can instruct the command to print all occurrences. The search pattern can be one word or a sequence of words.

Syntax

mem search [-addressradix <radix_type>] [-dataradix <radix_type>] [-all] [-replace <word>[<word>…]]
[-startaddress <address>] [-endaddress <address>] [<path>]
[-glob <word>[<word>…]] | [-regexp <word>[<word>…]]

Arguments

- -addressradix <radix_type>

  Specifies the radix for the address being displayed. The <radix_type> can be specified as decimal or hexadecimal. Default is decimal. Optional.

- -dataradix <radix_type>

  Specifies the radix for the memory data being displayed. The <radix_type> can be specified as symbolic, binary, octal, decimal, unsigned, or hex. Optional. By default the radix displayed is the system default.

- -all

  Searches the specified memory range and prints out all matching occurrences to the screen. Optional. By default only the first matching occurrence is printed.

- -replace <word>[<word>…]

  Replaces the found patterns with a designated pattern. Optional. If this option is used, each pattern specified by the **-pattern** argument must have a corresponding pattern specified by the **-replace** argument. Multiple word patterns are accepted, separated by a single white space. No wildcards are allowed in the replaced pattern.

- -startaddress <address>

  Specifies the start address for a range of addresses to search. The <address> can be specified as any valid address in the memory. Optional.

- -endaddress <address>

  Specifies the end address for a range of addresses to search. The <address> can be specified as any valid address in the memory. Optional.

- <path>

  Specifies the hierarchical path to the location of the memory instance. Optional. The default is the current **context** value, as shown in the Structure tab of the Workspace pane.

- -glob <word>[<word>…]

  Specifies the value of the pattern, accepting glob pattern syntax for the search. This argument and **-regexp** and **-pattern** are mutually exclusive arguments. This argument is functionally identical to the **-pattern** argument. Required: either **-glob** or **-regexp**.

  Multiple word patterns are accepted, separated by a single white space. Wildcards are accepted in the pattern.

- -regexp <word>[<word>…]

  Specifies the value of the pattern, accepting regular expression syntax, for the search. This argument and **-glob** and **-pattern** are mutually exclusive arguments. Required: either **-glob** or **-regexp**.

  Multiple word patterns are accepted, separated by a single white space. Wildcards are accepted in the pattern.

Examples

- Search for and print to the screen all occurrences of the pattern **16‘Hbeef** in */uut/u0/mem3*:

  ```
  mem search -glob 16‘Hbeef -dataradix hex /uut/u0/mem3
  ```

  Returns:

  ```
  #7845: beef
  #7846: beef
  #100223: beef
  ```

- Search for and print only the first occurrence of **16‘Hbeef** in the address range 7845:150000, replacing it with **16‘Hcafe** in */uut/u1/mem3*:

  ```
  mem search -glob 16‘Hbeef -d hex -replace 16‘Hcafe -st 7846 -end
  150000 /uut/u1/mem3
  ```

  Returns:

  ```
  #7846: cafe
  ```

- Replace all occurrences of **16‘Hbeef** with **16‘Habe** in */uut/u1/mem3*:

  ```
  mem search -glob 16‘Hbeef -r 16‘Habe -addressadix hex -all
  /uut/u1/mem3
  ```

  Returns:

  ```
  #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>
```

messages clearfilter

This command removes any filter you have set in the Message Viewer. Refer to the section "Message Viewer Filter Dialog Box" for additional information about filtering in the Message Viewer.

Syntax

messages clearfilter

Arguments

- No arguments

messages setfilter

This command performs the same action as the Message Viewer Filter Dialog Box, which controls which messages are shown in the Message Viewer.

The ideal workflow for using this command is through the GUI:

1. **View > Message Viewer**.
2. Right-click in the Message Viewer and select **Filter**.

   The Message Viewer Filter dialog box is displayed
3. Create your filter.
4. **OK** or **Apply**.

   The Message Viewer updates based on your filter and a messages setfilter command, which is equivalent to your settings, is output to the transcript.
5. Retain the messages setfilter command from the transcript for future use.

Syntax

messages setfilter <tcl_list>

Arguments

- <tcl_list> — The tcl_list argument is a complex string of tcl code that controls the filter settings.

Examples

- Severity is error and time is greater than or equal to 100 ns

```
messages setfilter {{} \
   ( Severity Contains {Case Insensitive} error )} \
   {AND ( Time >= 100 ns )}
```

- The objects field contains neither clock or reset

```
messages setfilter {{} \
   ( Object Contains {Case Sensitive} clock )} \
   {NOR ( Object Contains {Case Sensitive} data )}
```

- The message string either contains reg_str2 or reg_str1

```
messages setfilter {{} \
   ( Message Contains {Case Insensitive} reg_str2 )} \
   {OR ( Message Contains {Case Insensitive} reg_str1 )}
```

modelsim

The **modelsim** command starts the ModelSim GUI without prompting you to load a design.

This command is valid only for Windows platforms and may be invoked in one of three ways:

- from the DOS prompt
- from a ModelSim shortcut
- from the Windows Start > Run menu

To use **modelsim** arguments with a shortcut, add them to the target line of the shortcut's properties. (Arguments work on the DOS command line too, of course.)

The simulator may be invoked from either the ModelSim> prompt after the GUI starts or from a DO file called by **modelsim**.

Syntax

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

Arguments

- -do <macrofile>

  Specifies the DO file to execute when **modelsim** is invoked. Optional.

Note

In addition to the macro called by this argument, if a DO file is specified by the STARTUP variable in *modelsim.ini*, it will be called when the vsim command is invoked.

- -nosplash

  Disables the splash screen. Optional.

See also

vsim, do, "Using a Startup File"

noforce

The **noforce** command removes the effect of any active force commands on the selected HDL objects.

The **noforce** command also causes the object's value to be re-evaluated.

Syntax

noforce <object_name> …

Arguments

- <object_name>

  Specifies the name of an object. Required. Must match an object name used in a previous force command. Multiple object names may be specified. Wildcard characters are allowed.

See also

force and Wildcard Characters

nolog

The **nolog** command suspends writing of data to the wave log format (WLF) file for the specified signals.

A flag is written into the WLF file for each signal turned off, and the GUI displays "-No Data-" for the signal(s) until logging (for the signal(s)) is turned back on. Logging can be turned back on by issuing another log command or by doing a **nolog -reset**.

Because use of the **nolog** command adds new information to the WLF file, WLF files created when using the **nolog** command cannot be read by older versions of the simulator. If you are using *dumplog64.c*, you will need to get an updated version.

Syntax

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

Arguments

- -all

  Turns off logging for all signals currently logged. Optional.

- -depth <level>

  Restricts a recursive search (specified with the **-recursive** argument) to a certain level of hierarchy. <level> is an integer greater than or equal to zero. For example, if you specify -depth 1, the command descends only one level in the hierarchy. Optional.

- -howmany

  Returns an integer indicating the number of signals found. Optional.

- -in

  Turns off logging only for ports of mode IN whose names match the specification. Optional.

- -inout

  Turns off logging only for ports of mode INOUT whose names match the specification. Optional.

- -internal

  Turns off logging only for internal (non-port) objects whose names match the specification. Optional.

- -out

  Turns off logging only for ports of mode OUT whose names match the specification. Optional.

- -ports

  Specifies that the scope of the search is to include all ports. Optional.

- -recursive

  Specifies that the scope of the search is to descend recursively into subregions. Optional. 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.

- -reset

  Turns logging back on for all unlogged signals. Optional.

- <object_name>…

  Specifies the object name which you want to unlog. Optional. Multiple object names may be specified. Wildcard characters are allowed.

Examples

- Unlog all objects in the design.

  ```
  nolog -r /*
  ```

- Turn logging back on for all unlogged signals.

  ```
  nolog -reset
  ```

See also

add list, add wave, log

notepad

The **notepad** command opens a simple text editor. It may be used to view and edit ASCII files or create new files.

This mode can be changed from the Notepad Edit menu.

Returns nothing.

Syntax

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

Arguments

- <filename>

  Name of the file to be displayed. Optional.

- -r | -edit

  Selects the notepad editing mode: -r for read-only, and -edit for edit mode. Optional. Edit mode is the default.

noview

The **noview** command closes a window/pane in the ModelSim GUI. To open a window/pane, use the **view** command.

Syntax

noview [<class>] [<window_name>…]

Arguments

- <class>

  Specifies a class of windows in the MDI frame to close. All windows in that class will close. Valid values include: Source, List, Wave, and Memory. Optional.

- <window_name>…

  Specifies the window/pane to close. Wildcards and multiple window/pane types may be used. At least one type (or wildcard) is required. Available window types are:

  ```
  dataflow, list, locals, memory, objects, process, profilemain,
  profiledetails, signals, structure, variables, wave, watch, and
  workspace
  ```

  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
  ```

See also

view

nowhen

The **nowhen** command deactivates selected when commands.

Syntax

nowhen [<label>]

Arguments

- <label>

  Specifies an individual when command. Optional. Wildcards may be used to select more than one when command.

Examples

- This **nowhen** command deactivates the when command labeled 99.

```
when -label 99 b {echo “b changed”}
…
nowhen 99
```

- This **nowhen** command deactivates all when commands.

```
nowhen *
```

onbreak

The **onbreak** command is used within a macro, which must be followed by a **run** command to take effect. It specifies one or more commands to be executed when running a macro that encounters a breakpoint in the source code.

Using the **onbreak** command without arguments will return the current **onbreak** command string. An **onbreak** command can contain macro calls.

The default behavior for the onbreak command is the resume command.

Use an empty string to change the **onbreak** command back to its default behavior:

```
onbreak ""
```

In this case, the macro will be interrupted after a breakpoint occurs (after any associated bp command string is executed).

Syntax

onbreak {[<command> [; <command>] …]}

Arguments

- <command>

  Any command can be used as an argument to **onbreak**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces. You must use the onbreak command before a run, **run -continue**, or step command. It is an error to execute any commands within an **onbreak** command string following any of the run commands. This restriction applies to any macros or Tcl procedures used in the **onbreak** command string. Optional.

Examples

- Examine the value of the HDL object data when a breakpoint is encountered. Then continue the run command.

  ```
  onbreak {exa data ; cont}
  ```

- Resume execution of the macro file on encountering a breakpoint.

  ```
  onbreak {resume}
  ```

- This set of commands test for assertions. Assertions are treated as breakpoints if the severity level is greater than or equal to the current BreakOnAssertion variable setting (refer to “Simulator Control Variables”). By default a severity level of failure or above causes a breakpoint; a severity level of error or below does not.

```
set broken 0
onbreak {
  set broken 1
  resume
}
run -all
if { $broken } {
  puts "failure"
} else {

  puts "success"
}
```

See also

abort, bd, bp, do, onerror, resume, status

onElabError

The **onElabError** command specifies one or more commands to be executed when an error is encountered during the elaboration portion of a **vsim** command. The command is used by placing it within a macro.

Use the **onElabError** command without arguments to return to a prompt.

Syntax

onElabError {[<command> [; <command>] …]}

Arguments

- <command>

  Any command can be used as an argument to **onElabError**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces. Optional.

See also

do

onerror

The **onerror** command is used within a macro, placed before a **run** command; it specifies one or more commands to be executed when a running macro encounters an error.

Using the **onerror** command without arguments will return the current **onerror** command string. Use an empty string to change the **onerror** command back to its default behavior (i.e., onerror ""). Use **onerror** with a resume command to allow an error message to be printed without halting the execution of the macro file.

You can also set the global OnErrorDefaultAction Tcl variable to dictate what action ModelSim takes when an error occurs. To set the variable on a permanent basis, you must define the variable in a *modelsim.tcl* file (Refer to "The modelsim.tcl File" for details).

The **onerror** command is executed when a Tcl command encounters an error in the macro file that contains the **onerror** command (note that a **run** command does not necessarily need to be in process). Conversely, OnErrorDefaultAction will run even if the macro does not contain a local **onerror** command. This can be useful when you run a series of macros from one script, and you want the same behavior across all macros.

Syntax

onerror {[<command> [; <command>] …]}

Arguments

- <command>

  Any command can be used as an argument to **onerror**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces. Optional.

Example

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

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

See also

abort, do, onbreak, resume, status

pause

The **pause** command placed within a macro interrupts the execution of that macro, allowing you to perform interactive debugging of the macro file.

Syntax

pause

Arguments

- None.

Description

When you execute a macro and that macro gets interrupted, the prompt will change to:

```
VSIM(paused)>
```

This “pause” prompt reminds you that a macro has been interrupted.

When a macro is paused, you may invoke another macro, and if that one gets interrupted, you may even invoke another — up to a nesting level of 50 macros.

If the status of nested macros gets confusing, use the status command. It will show you which macros are interrupted, at what line number, and show you the interrupted command.

To resume the execution of the macro, use the resume command. To abort the execution of a macro use the abort command.

See also

abort, do, resume, run, status

precision

The **precision** command determines how real numbers display in the graphic interface (e.g., Objects, Wave, Locals, and List windows). It does not affect the internal representation of a real number and therefore precision values over 17 are not allowed.

Using the **precision** command without any arguments displays the current precision setting.

Syntax

precision [<digits>[#]]

Arguments

- <digits>[#]

  Specifies the number of digits to display. Optional. Default is 6. Trailing zeros are not displayed unless you append the '#' sign. See examples for more details.

Examples

- Results in 4 digits of precision.

  ```
  precision 4
  ```

  For example:

  ```
  1.234 or 6543
  ```

- Results in 8 digits of precision including trailing zeros.

  ```
  precision 8#
  ```

  For example:

  ```
  1.2345600 or 6543.2100
  ```

- Results in 8 digits of precision but doesn't print trailing zeros.

  ```
  precision 8
  ```

  For example:

  ```
  1.23456 or 6543.21
  ```

printenv

The **printenv** command prints to the Transcript pane the current names and values of all environment variables.

If variable names are given as arguments, prints only the names and values of the specified variables.

Syntax

printenv [<var>…]

Arguments

- <var>…

  Specifies the name(s) of the environment variable(s) to print. Optional.

Examples

- Print all environment variable names and their current values.

  ```
  printenv
  ```

  For example,

  ```
  # CC = gcc
  # DISPLAY = srl:0.0
  …
  ```

- Print the specified environment variables:

  ```
  printenv USER HOME

  # USER = vince
  # HOME = /scratch/srl/vince
  ```

project

The **project** command is used to perform common operations on projects. Some of the project commands must be used outside of a simulation session.

Syntax

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

Arguments

- addfile <filename> [<file_type>] [<folder_name>]

  Adds the specified file to the current open project. Optional. You may also specifiy the HDL file type and folder name in which the file will be placed. If no folder name is specified the file will be placed in the top level folder.

- addfolder <foldername> [<folder_parent>]

  Creates a folder to contain the project. Optional. You may also specify a parent folder for the project folder. If unspecified, the project folder is placed at the top level.

- calculateorder

  Determines the compile order for the project by compiling each file, then moving any compiles that fail to the end of the list. This is repeated until there are no more compile errors. Optional.

- close

  Closes the current project. Optional.

- compileall [-n]

  Compiles all files in the project using the defined compile order. Optional. The -n option will return a list of the compile commands this command would execute, without actually executing the compiles.

- compileorder

  Returns the current compile order list. Optional.

- compileoutofdate [-n]

  Compiles all files that have a newer date/time stamp than the last time the file was compiled. Optional. The -n option will return a list of the compile commands this command would execute, without actually executing the compiles.

- delete <filename>

  Deletes the specified project (*.mpf*) file. Optional.

- env

  Returns the current project file. Optional.

- history

  Lists a history of manipulated projects. Optional. Must be used outside of a simulation session.

- new <home_dir> <proj_name> [<defaultlibrary>] [<intialini>] [<reference>]

  Creates a new project under a specified home directory with a specified name and optionally a default library. Optional. The name of the work library will default to "work" unless specified. An optional *modelsim.ini* file can be specified as a seed for the project file by using the initialini option. If initialini is an empty string, then ModelSim uses the current *modelsim.ini* file when creating the project. You must specify a default library if you want to specify initialini. A new project cannot be created while a project is currently open or a simulation is in progress. The boolean "reference" option indicates if library mappings will include an "others" clause back to the initial *.ini* file (1) or copy all the mappings into the new file (0).

- open <project>

  Closes any currently opened project and opens a specified project file (must be a valid *.mpf* file), making it the current project. Changes the current working directory to the project's directory. Optional. Must be used outside of a simulation session.

- removefile <filename>

  Removes the specified file from the current project. Optional.

Examples

- Make */user/george/design/test3/test3.mpf* the current project and changes the current working directory to */user/george/design/test3*.

  ```
  project open /user/george/design/test3/test3.mpf
  ```

- Execute current project library build scripts.

  ```
  project compileall
  ```

pwd

The Tcl **pwd** command displays the current directory path in the Transcript pane.

Syntax

pwd

Arguments

- None

quietly

The **quietly** command turns off transcript echoing for the specified command.

Syntax

quietly <command>

Arguments

- <command>

  Specifies the command for which to disable transcript echoing. Required. Any results normally echoed by the specified command will not be written to the Transcript pane. To disable echoing for all commands use the transcript command with the **-quietly** option.

See also

transcript

quit

The **quit** command exits the simulator.

If you want to stop the simulation using a when command, you must use a stop command within your when statement, you must not use an exit or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated.

Syntax

quit [-f | -force] [-sim] [-code <integer>]

Arguments

- -f | -force

  Quits without asking for confirmation. Optional. If omitted, ModelSim asks you for confirmation before exiting. (The -f and -force arguments are equivalent.)

- -sim

  Unloads the current design in the simulator without exiting ModelSim. All files opened by the simulation will be closed including the WLF file (*vsim.wlf*).

- -code <integer>

  Quits the simulation and issues an exit code.

  <integer> — This is the value of the exit code. You should not specify an exit code that already exists in the tool. Refer to the section "Exit Codes" in the User's Manual for a list of existing exit codes. You can also specify a variable in place of the <integer>.

  You should always print a message before executing the quit -code command to explicitly state the reason for exiting.

Examples

Refer to the Examples section of the exit command for an example of using the -code argument. The quit and exit commands behave similarly in this regard.

radix

The **radix** command specifies the default radix to be used for the current simulation.

The command can be used at any time. The specified radix is used for all commands (force, examine, change, etc.) as well as for displayed values in the Objects, Locals, Dataflow, List, and Wave windows.

Alternate methods for changing the default radix:

- In the *modelsim.ini* file, edit the DefaultRadix variable.
- Choose **Simulate > Runtime Options** from the main menu, click the **Defaults** tab, make your selection in the **Default Radix** box.

Syntax

radix [-symbolic | -binary | -octal | -decimal | -hexadecimal | -unsigned | -ascii | -time]

Arguments

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

- -symbolic

  Displays values in a form closest to their natural form. Optional.
- -binary

  Displays values in binary format. Optional.
- -octal

  Displays values in octal format. Optional.
- -decimal

  Displays values in decimal format. You can specify -signed as an alias for this argument. Optional.
- -hexadecimal

  Displays values in hexadecimal format. Optional.
- -unsigned

  Displays values in unsigned decimal format. Optional.
- -ascii

  Display a Verilog object as a string equivalent using 8-bit character encoding. Optional.
- -time

  Displays values of time for register-based types in Verilog. Optional.
- <no argument>

  Returns the current radix setting.

See also

User-Defined Radices, radix define, radix names, radix list, radix delete

radix define

The **radix define** command is used to create or modify a user-defined radix. A user definable radix is used to map bit patterns to a set of enumeration labels. User-defined radices are available for use in the Wave and List windows or with the examine command.

Syntax

radix define <name> <definition_body>

Arguments

- <name>

  User-specified name for the radix. Required.

- <definition_body>

  A list of number pattern, label pairs. Required. The definition body has the form:

```
{
    <numeric-value> <enum-label>,
    <numeric-value> <enum-label>
    -default <radix_type>
}
```

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

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

  Check the Verilog and VHDL LRM's for exact definitions of these numeric literals.

  The comma (,) in the definition body is optional. The <enum-label> is any arbitrary string. It should be quoted ("") especially if it contains spaces.

  The -default entry is optional. If present, it defines the radix to use if a match is not found for a given value. The -default entry can appear anywhere in the list, it does not have to be at the end.

Example

- The radix define command used to create a radix called “States,” which will display state values in the List, Watch, and Wave windows instead of numeric values.

```
radix define States {
    11'b00000000001 "IDLE",
    11'b00000000010 "CTRL",
    11'b00000000100 "WT_WD_1",
    11'b00000001000 "WT_WD_2",
    11'b00000010000 "WT_BLK_1",
    11'b00000100000 "WT_BLK_2",
    11'b00001000000 "WT_BLK_3",
    11'b00010000000 "WT_BLK_4",
    11'b00100000000 "WT_BLK_5",
```

```
        11'b01000000000 "RD_WD_1",
        11'b10000000000 "RD_WD_2",
        -default hex
    }
```

See also

User-Defined Radices, radix, radix names, radix list, radix delete

radix names

The **radix names** command returns a list of currently defined radix names.

Syntax

radix name

Arguments

None

See also

User-Defined Radices, radix, radix define, radix list, radix delete

radix list

The **radix list** command will return the complete definition of a radix, if a name is given. If no name is given, it will list all the defined radices.

Syntax

radix list [<name>]

Arguments

- <name>

  Returns the complete definition of the named radix. Optional.

See also

User-Defined Radices, radix, radix define, radix names, radix delete

radix delete

The **radix delete** command will remove the radix definition from the named radix.

Syntax

radix delete <name>

Arguments

- <name>

  Removes the radix definition from the named radix. Required.

See also

User-Defined Radices, radix, radix define, radix names, radix list

readers

The **readers** command displays the names of all readers of the specified object.

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

Syntax

readers <object_name>

Arguments

- <object_name>

  Specifies the name of the signal or net whose readers are to be shown. Required. All signal or net types are valid. Multiple names and wildcards are accepted.

See also

drivers

report

The **report** command displays the value of all simulator control variables, or the value of any simulator state variables relevant to the current simulation.

Syntax

report simulator control | simulator state

Arguments

- 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 all simulator control variables.

```
report simulator control

# UserTimeUnit = ns
# RunLength =
# IterationLimit = 5000
# BreakOnAssertion = 3
# DefaultForceKind = default
# IgnoreNote = 0
# IgnoreWarning = 0
# IgnoreError = 0
# IgnoreFailure = 0
# IgnoreSVAInfo= 1
# IgnoreSVAWarning = 1
# 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. Only the variables that relate to the design being simulated are displayed:

```
report simulator state
```

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

Viewing preference variables

Preference variables have more to do with the way things look (but not entirely) rather than controlling the simulator. You can view preference variables from the Preferences dialog box. Select **Tools > Edit Preferences** (Main window).

See also

Simulator Control Variables, Simulator GUI Preferences

restart

The **restart** 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.)

Shared libraries are handled as follows during a restart:

- Shared libraries that implement VHDL foreign architectures only are reloaded at each restart when the architecture is elaborated (unless the **-keeploaded** option to the vsim command 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 you specify the **-keeploaded** argument to **vsim**).
- Shared libraries that implement VHDL foreign subprograms remain loaded (they are not reloaded) even if they also contain code for a foreign architecture.

You can configure defaults for the restart command by setting the **DefaultRestartOptions** variable in the *modelsim.ini* file. Refer to "Restart Command Defaults".

To handle restarts with Verilog PLI applications, you need to define a Verilog user-defined task or function, and register a misctf class of callback. To handle restarts with Verilog VPI applications, you need to register reset callbacks. To handle restarts with VHDL FLI applications, you need to register restart callbacks. Refer to "Verilog PLI/VPI/DPI" for more information on the Verilog PLI/VPI/DPI and the *ModelSim FLI Reference* for more information on the FLI.

Syntax

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

Arguments

- -force

  Specifies that the simulation will be restarted without requiring confirmation in a popup window. Optional.

- -nobreakpoint

  Specifies that all breakpoints will be removed when the simulation is restarted. Optional. The default is for all breakpoints to be reinstalled after the simulation is restarted.

- -nolist

  Specifies that the current List window environment will **not** be maintained after the simulation is restarted. Optional. The default is for all currently listed HDL objects and their formats to be maintained.

- -nolog

  Specifies that the current logging environment will **not** be maintained after the simulation is restarted. Optional. The default is for all currently logged objects to continue to be logged.

- -nowave

  Specifies that the current Wave window environment will **not** be maintained after the simulation is restarted. Optional. The default is for all objects displayed in the Wave window to remain in the window with the same format.

See also

vsim

resume

The **resume** command is used to resume execution of a macro file after a pause command or a breakpoint.

It may be input manually or placed in an onbreak command string. (Placing a **resume** command in a bp command string does not have this effect.) The **resume** command can also be used in an onerror command string to allow an error message to be printed without halting the execution of the macro file.

Syntax

resume

Arguments

- None

See also

abort, do, onbreak, onerror, pause

run

The **run** command advances the simulation by the specified number of timesteps.

Syntax

run [<timesteps>[<time_units>]] | [-all] | [-continue] | [-next] | [-step] | [-over]

Arguments

- <timesteps>[<time_units>]

  Specifies the number of timesteps for the simulation to run. The number may be fractional, or may be specified absolute by preceding the value with the character @. Optional. In addition, optional <time_units> may be specified as:

  ```
  fs, ps, ns, us, ms, or sec
  ```

  The default <timesteps> and <time_units> specifications can be changed during a ModelSim session by selecting **Simulate > Simulation Options** (Main window). Time steps and time units may also be set with the RunLength and UserTimeUnit variables in the *modelsim.ini* file.

- -all

  Causes the simulator to run the current simulation forever, or until it hits a breakpoint or specified break event. Optional.

- -continue

  Continues the last simulation run after a step command, **step -over** command or a breakpoint. A **run -continue** command may be input manually or used as the last command in a bp command string. Optional.

- -next

  Causes the simulator to run to the next event time. Optional.

- -step

  Steps the simulator to the next HDL statement. Optional.

- -over

  Specifies that VHDL procedures, functions and Verilog tasks are to be executed but treated as simple statements instead of entered and traced line by line. Optional.

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
```

See also

step

runStatus

The runStatus command returns the current state of your simulation after issuing a run or step command.

Syntax

runStatus [-full]

Arguments

- -full

  appends additional information to the output of the runStatus command.

Results

The output of the runStatus command is described in Table 2-2 (runStatus results) and Table 2-3 (runStatus -full results).

Table 2-2. 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 2-3. 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 2-3. runStatus -full Command Information

-full Information	Description
step_wait_suspend	run -step completed, time advanced.
user_break	run interrupted do to break-key or ^C (SIGINT)
user_halt	mti_Break() called.
user_stop	stop or finish requested from vpi, stop command, etc.
gate_oscillation	Verilog gate iteration limit reached.
simulation_stop	pli stop_simulation() called.

Disables the generation of symbols for the debugging database in the library, which allows source annotation.

searchlog

The **searchlog** command searches one or more of the currently open logfiles for a specified condition.

It can be used to search for rising or falling edges, for signals equal to a specified value, or for when a generalized expression becomes true.

Syntax

searchlog [-count <n>] [-deltas] [-env <path>] [-expr {<expr>}] [-reverse]
[-rising | -falling | -anyedge] [-startDelta <num>] [-value <string>] <startTime> <pattern>

Description

If at least one match is found, it returns the time (and optionally delta) at which the last match occurred and the number of matches found, in a Tcl list:

```
{{<time>} <matchCount>}
```

where **<time>** is in the format **<number>** **<unit>**. If the **-deltas** option is specified, the delta of the last match is also returned:

```
{{<time>} <delta> <matchCount>}
```

If no matches are found, a TCL_ERROR is returned. If one or more matches are found, but less than the number requested, it is not considered an error condition, and the time of the farthest match is returned, with the count of the matches found.

Arguments

- -count <n>

  Specifies to search for the nth occurrence of the match condition, where <n> is a positive integer. Optional.

- -deltas

  Indicates to test for a match on simulation delta cycles. Otherwise, matches are only tested for at the end of each simulation time step. Optional.

- -env <path>

  Provides a design region in which to look for the signal names. Optional.

- -expr {<expr>}

  Specifies a general expression of signal values and simulation time. Optional. **searchlog** will search until the expression evaluates to true. The expression must have a boolean result type. See GUI_expression_format for the format of the expression.

- -reverse

  Specifies to search backwards in time from <startTime>. Optional.

- -rising | -falling | -anyedge

  Specifies an edge to look for on a scalar signal. Optional. This option is ignored for compound signals. If no options are specified, the default is -anyedge.

- -startDelta <num>

  Indicates a simulation delta cycle on which to start. Optional.

- -value <string>

  Specifies to search until a single scalar or compound signal takes on this value. Optional.

- <startTime>

  Specifies the simulation time at which to start the search. Required. The time may be specified as an integer number of simulation units, or as {<num> <timeUnit>}, where <num> can be integer or with a decimal point, and <timeUnit> is one of the standard VHDL time units (fs, ps, ns, us, ms, sec).

- <pattern>

  Specifies one or more signal names or wildcard patterns of signal names to search on. Required unless the **-expr** argument is used.

See also

virtual signal, virtual log, virtual nolog

see

The see command displays the specified number of source file lines around the current execution line. By default, five lines will be displayed before and four lines after.

Syntax

see [<n> | <pre> <post>]

Arguments

- <n>

  Designates the number of lines to display before and after the current execution line. Optional.

- <pre>

  Designates the number of lines to display before the current execution line. Optional.

- <post>

  Designates the number of lines to display after the current execution line. Optional.

Example

- Display 8 lines before and 6 lines after the current execution line.

  ```
  see 8 6
  ```

setenv

The **setenv** command changes or reports the current value of an environment variable. The setting is not persistent–it is valid only for the current ModelSim session.

Syntax

setenv <varname> [<value>]

Arguments

- <varname>

  The name of the environment variable you wish to set or check. Required.

- <value>

  The value for the environment variable. Optional. If you don't specify a value, ModelSim reports the variable's current value.

See also

unsetenv, printenv

shift

The **shift** 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, etc. The previous value of \$1 is discarded.

The **shift** command and macro parameters are used in macro files. If a macro file requires more than nine parameters, they can be accessed using the **shift** command.

To determine the current number of macro parameters, use the argc variable.

Syntax

shift

Arguments

- None

Description

For a macro file containing nine macro parameters defined as $1 to $9, one **shift** command shifts all parameter values one place to the left. If more than nine parameters are named, the value of the tenth parameter becomes the value of $9 and can be accessed from within the macro file.

See also

do

show

The **show** command lists HDL objects and subregions visible from the current environment.

The objects listed include:

- **VHDL** — signals, processes, constants, variables, and instances
- **Verilog —** nets, registers, tasks, functions, instances, variables, and memories

The **show** command returns formatted results to stdout. To eliminate formatting (to use the output in a Tcl script), use the **Show** command instead.

Syntax

show [-all] [<pathname>]

Arguments

- -all

  Displays all names at and below the specified path recursively. Optional.

- <pathname>

  Specifies the pathname of the environment for which you want the objects and subregions to be listed. Optional; if omitted, the current environment is assumed.

Examples

- List the names of all the objects and subregion environments visible in the current environment.

  ```
  show
  ```

- List the names of all the objects and subregions visible in the environment named /uut.

  ```
  show /uut
  ```

- List the names of all the objects and subregions visible in the environment named sub_region which is directly visible in the current environment.

  ```
  show sub_region
  ```

See also

find

simstats

The **simstats** command returns performance-related statistics about elaboration and simulation. The statistics measure the simulation kernal process (vsimk) for a single invocation of vsim. If you invoke vsim a second time, or restart the simulation, the current statistics are discarded and new values are collected.

If executed without arguments, the command returns a list of pairs like the following:

```
{{elab memory} 0} {{elab working set} 7245824} {{elab time} 0.942645}
{{elab cpu time} 0.190274} {{elab context} 0} {{elab page faults} 1549}
{memory 0} {{working set} 0} {time 0} {{cpu time} 0} {context 0}
{{page faults} 0}
```

The elaboration statistics are measured one time at the end of elaboration. The simulation memory statistics are measured at the time you invoke **simstats**. The simulation time statistics are updated at the end of each run command. See the arguments below for descriptions of each statistic.

Units for time values are in seconds. Units for memory values vary by platform:

- For SunOS and Linux, the memory size is reported in Kbytes
- For Windows, the memory size is reported in bytes.

Some of the values may not be available on all platforms and other values may be approximates. Different operating systems report these numbers differently.

Syntax

simstats [memory | working | time | cpu | context | faults]

Arguments

- memory

  Returns the amount of virtual memory that the OS has allocated for vsimk. Optional.

- working

  Returns the portion of allocated virtual memory that is currently being used by vsimk. Optional. If this number exceeds the actual memory size, you will encounter performance degradation.

- time

  Returns the cumulative "wall clock time" of all run commands. Optional.

- cpu

  Returns the cumulative processor time of all run commands. Optional. Processor time differs from wall clock time in that processor time is only counted when the cpu is actually running vsimk. If vsimk is swapped out for another process, cpu time does not increase.

- context

  Returns the number of context swaps (vsimk being swapped out for another process) that occurred during all run commands. Optional.

- faults

  Returns the number of page faults that occurred during all run commands. Optional.

status

The **status** command lists summary information about currently interrupted macros.

If invoked without arguments, the command lists the filename of each interrupted macro, the line number at which it was interrupted, and prints the command itself. It also displays any onbreak or onerror commands that have been defined for each interrupted macro.

Syntax

status [file | line]

Arguments

- file

  Reports the file pathname of the current macro.

- line

  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
```

See also

abort, do, pause, resume

step

The **step** command steps to the next HDL or C statement. Current values of local HDL variables may be observed at this time using the Locals window.

VHDL procedures and functions, Verilog tasks and functions, and C functions can optionally be skipped over. When a wait statement or end of process is encountered, time advances to the next scheduled activity. The Process and Source windows will then be updated to reflect the next activity.

Syntax

step [-over] [<n>]

Arguments

- -over

  Specifies that VHDL procedures and functions, Verilog tasks and functions, and C functions should be executed but treated as simple statements instead of entered and traced line by line. Optional.

- <n>

  Any integer. Optional. Will execute ‘n’ steps before returning.

See also

run

stop

The **stop** command is used with the when command to stop simulation in batch files.

The **stop** command has the same effect as hitting a breakpoint. The **stop** command may be placed anywhere within the body of the when command.

Syntax

stop

Arguments

- None.

Description

Use the run command with the **-continue** option to continue the simulation run, or the resume command to continue macro execution. If you want macro execution to resume automatically, put the **resume** command at the top of your macro file:

```
onbreak {resume}
```

Note

If you want to stop the simulation using a when command, you must use a stop command within your when statement. DO NOT use an exit command or a quit command. The **stop** command acts like a breakpoint at the time it is evaluated.

See also

bp, resume, run, when

suppress

The **suppress** command prevents the specified message(s) from displaying. You cannot suppress Fatal or Internal messages. The **suppress** command used without arguments will return the message numbers of all suppressed messages.

Edit the suppress variable in the modelsim.ini file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

Syntax

suppress [-clear <msg_number>[,<msg_number>,...]] [<msg_number>[,<msg_number>,…]]

Arguments

- -clear <msg_number>[,<msg_number>,...]

  Clears message suppression of the message (or messages) designated by the message number (or numbers). Optional.

- <msg_number>[,<msg_number>,...]

  The number (or numbers) preceding the message (or messages) you wish to suppress. At least one message number is required. Optional.

Examples

- Return the message numbers of all suppressed messages:

  ```
  suppress
  ```

- Suppress designated messages:

  ```
  suppress 8241,8242,8243,8446,8447
  ```

- Clear message suppression for the designated messages:

  ```
  suppress -clear 8241,8242
  ```

- Return the message numbers of all suppressed messages and clear suppression for all:

  ```
  suppress -clear suppress
  ```

tb

The **tb** (traceback) command displays a stack trace for the current process in the Transcript pane. This lists the sequence of HDL function calls that have been entered to arrive at the current state for the active process.

Syntax

tb

Arguments

- None

Time

There are several Time commands that allow you to perform comparisons between, operations on, and conversions of time values.

Syntax

eqTime <time1> <time2>

Returns a 1 (true) or 0 (false) if <time1> and <time2> are equal.

neqTime <time1> <time2>

Returns a 1 (true) or 0 (false) if <time1> and <time2> are not equal.

ltTime <time1> <time2>

Returns a 1 (true) or 0 (false) if <time1> is less than <time2>.

gtTime <time1> <time2>

Returns a 1 (true) or 0 (false) if <time1> is greater than <time2>.

lteTime <time1> <time2>

Returns a 1 (true) or 0 (false) if <time1> is less than or equal to <time2>.

gteTime <time1> <time2>

Returns a 1 (true) or 0 (false) if <time1> is greater than or equal to <time2>.

addTime <time1> <time2>

Returns the value of adding <time1> to <time2>

subTime <time1> <time2>

Returns the value of subtracting <time2> from <time1>

mulTime <time1> <integer>

Returns the value of multiplying <time1> by an <integer>

divTime <time1> <time2>

Returns an integer, which is the value of dividing <time1> by <time2>. Specifying 0 for <time2> results in an error.

intToTime <high_32bit_int> <low_32bit_int>

Returns a 64-bit time value based on two 32-bit parts of a 64-bit integer. This command is useful when you've performed an integer calculation that results in a 64-bit value and need to convert it to a time unit.

scaleTime <time1> <scale_factor>

Returns a time value scaled by a real number and truncated to the current time resolution.

RealToTime <real>

Returns a time value equivalent to the specified real number and truncated to the current time resolution.

validTime <time>

Returns a 1 (true) or 0 (false) if the given string is a valid time for use with any of these Time calculations.

formatTime {+ | -} commas | {+ | -}nodefunit | {+ | -}bestunits

Sets display properties for time values.

Arguments

- <time> —

  <number> — the command assumes that the <time_unit> is the current simulation time unit, as defined by the Resolution variable in the modelsim.ini file or the -t switch to the vsim command.

  <number><time_unit> — note that there is no space is between the values.

  <number> <time_unit> — note that if you put a space between the values, you must enclose the argument in braces ({ }) or double-quotes (" ").

- <time_unit> —

  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)

- <high_32bit_int> | <low_32bit_int>

  <high_32bit_int> — The "high" part of the 64-bit integer.

  <low_32bit_int> — The "low" part of the 64-bit integer.

- <scale_factor> — a real number to be used as scaling factor. Common values can include:

  ```
  0.25, 0.5, 1.5, 2, 10, 100
  ```

- {+ | -} commas — controls whether commas are displayed in time values.

  +commas — time values include commas

  -commas — time values do not include commas

- {+ | -}nodefunit — controls whether time values display time units

  +nodefunit — time values do not include time units and will be in current time resolution

-nodefunit — time values may include time units

- {+ | -}bestunits — controls whether time values display the largest possible time unit, for example 8 us instead of 8,000 ns.

  +bestunits — time values display the largest possible time unit

  -bestunits — time values display the default time unit

Examples

- The following transcript shows examples of the Time commands and their output:

```
>ltTime 100ns 1ms
# 1

>addTime {1545 ns} {455 ns}
# 2 us

>gteTime "1000 ns" "1 us"
# 1

>divTime 1us 10ns
# 100

>formatTime +bestunit
>scaleTime 3ms 1000
# 3 sec

>RealToTime 1.345e04
# 13450 ns
```

transcript

The **transcript** command controls echoing of commands executed in a macro file.

If no option is specified, the current setting is reported.

Syntax

transcript [on | off | -q | quietly]

Arguments

- on

  Specifies that commands in a macro file will be echoed to the Transcript pane as they are executed. Optional.

- off

  Specifies that commands in a macro file will not be echoed to the Transcript pane as they are executed. Optional.

- -q

  Returns "0" if transcripting is turned off or "1" if transcripting is turned on. Useful in a Tcl conditional expression. Optional.

- quietly

  Turns off the transcript echo for all commands. To turn off echoing for individual commands see the quietly command. Optional.

Examples

- Commands within a macro file will be echoed to the Transcript pane as they are executed.

  ```
  transcript on
  ```

- If issued immediately after the previous example, the message:

  ```
  transcript

  Macro transcripting is turned ON.
  ```

  appears in the Transcript pane.

See also

Transcript Window, echo

transcript file

The **transcript file** command sets or queries the pathname for the transcript file. 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.

Syntax

transcript file [<filename>]

Arguments

- <filename>

  Specifies the full path and filename for the transcript file. Optional. If you specify a new file, the existing transcript file is closed and a new transcript file opened. If you specify an empty string (""), the existing file is closed and no new file is opened. If you don’t specify this argument, the current setting is returned.

Examples

- Close the current transcript file and stops writing data to the file. This is a method for reducing the size of your transcript.

  ```
  transcript file ""
  ```

- This series of commands results in the transcript containing only data from the second millisecond of the simulation. The first **transcript file** command closes the transcript so no data is being written to it. The second **transcript file** command opens a new transcript and records data from 1 ms to 2 ms.

  ```
  transcript file ""
  run 1 ms
  transcript file transcript
  run 1 ms
  ```

See also

Transcript Window

tssi2mti

The **tssi2mti** command is used to convert a vector file in TSSI Format into a sequence of force and run commands.

The stimulus is written to the standard output.

The source code for **tssi2mti** is provided in the file *tssi2mti.c* in the *examples* directory.

Syntax

tssi2mti <signal_definition_file> [<sef_vector_file>]

Arguments

- <signal_definition_file>

  Specifies the name of the TSSI signal definition file describing the format and content of the vectors. Required.

- <sef_vector_file>

  Specifies the name of the file containing vectors to be converted. If none is specified, standard input is used. Optional.

Examples

- The command will produce a do file named *trigger.do* from the signal definition file *trigger.def* and the vector file *trigger.sef*.

  ```
  tssi2mti trigger.def trigger.sef > trigger.do
  ```

- This example is the same as the previous one, but uses the standard input instead.

  ```
  tssi2mti trigger.def < trigger.sef > trigger.do
  ```

See also

force, run, write tssi

unsetenv

The **unsetenv** command deletes an environment variable. The deletion is not permanent–it is valid only for the current ModelSim session.

Syntax

unsetenv <varname>

Arguments

- <varname>

  The name of the environment variable you wish to delete. Required.

See also

setenv, printenv

vcd add

The **vcd add** command adds the specified objects to a VCD file.

The allowed objects are Verilog nets and variables and VHDL signals of type bit, bit_vector, std_logic, and std_logic_vector (other types are silently ignored). The command works with mixed HDL designs.

All **vcd add** commands must be executed at the same simulation time. The specified objects are added to the VCD header and their subsequent value changes are recorded in the specified VCD file. By default all port driver changes and internal variable changes are captured in the file. You can filter the output using arguments detailed below.

Related Verilog tasks: $dumpvars, $fdumpvars

Syntax

vcd add [-r] [-in] [-out] [-inout] [-internal] [-ports] [-file <filename>] [-dumpports] <object_name> ...

Arguments

- -r

  Specifies that signal and port selection occurs recursively into subregions. Optional. If omitted, included signals and ports are limited to the current region.

- -in

  Includes only port driver changes from ports of mode IN. Optional.

- -out

  Includes only port driver changes from ports of mode OUT. Optional.

- -inout

  Includes only port driver changes from ports of mode INOUT. Optional.

- -internal

  Includes only internal variable or signal changes. Excludes port driver changes. Optional.

- -ports

  Includes only port driver changes. Excludes internal variable or signal changes. Optional.

- -file <filename>

  Specifies the name of the VCD file. This option should be used only when you have created multiple VCD files using the vcd files command.

- -dumpports

  Specifies port driver changes to be added to an extended VCD file. Optional. When the **vcd dumpports** command cannot specify all port driver changes that will appear within the VCD file, multiple **vcd add -dumpports** commands can be used to specify additional port driver changes.

- <object_name> ...

  Specifies the Verilog or VHDL object or objects to add to the VCD file. Required. Multiple objects may be specified by separating names with spaces. Wildcards are accepted.

See also

"Value Change Dump (VCD) Files". Verilog tasks are documented in the IEEE 1364 standard.

vcd checkpoint

The **vcd checkpoint** command dumps the current values of all VCD variables to the specified VCD file. While simulating, only value changes are dumped.

Related Verilog tasks: $dumpall, $fdumpall

Syntax

vcd checkpoint [<filename>]

Arguments

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the vcd file command or "dump.vcd" if **vcd file** was not invoked.

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd comment

The **vcd comment** command inserts the specified comment in the specified VCD file.

Syntax

vcd comment <comment string> [<filename>]

Arguments

- <comment string>

  Comment to be included in the VCD file. Required. Must be quoted by double quotation marks or curly braces.

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the vcd file command or "dump.vcd" if **vcd file** was not invoked.

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpports

The **vcd dumpports** command creates a VCD file that includes port driver data.

By default all port driver changes are captured in the file. You can filter the output using arguments detailed below. Related Verilog task: $dumpports

Syntax

vcd dumpports [-compress] [-direction] [-file <filename>] [-in] [-inout] [-out] [-no_strength_range] [-unique] [-vcdstim] <object_name> ...

Arguments

- -compress

  Produces a compressed VCD file. Optional. ModelSim uses the gzip compression algorithm. If you specify a *.gz* extension on the **-file <filename>** argument, ModelSim compresses the file even if you don't use the **-compress** argument.

- -direction

  Includes driver direction data in the VCD file. Optional.

- -file <filename>

  Specifies the path and name of a VCD file to create. Optional. Defaults to the current working directory and the filename *dumpports.vcd*. Multiple filenames can be opened during a single simulation.

- -in

  Includes ports of mode IN. Optional.

- -inout

  Includes ports of mode INOUT. Optional.

- -out

  Includes ports of mode OUT. Optional.

- -no_strength_range

  Ignores strength ranges when resolving driver values. Optional. This argument is an extension to the IEEE 1364 specification. Refer to "Resolving Values" for additional information.

- -unique

  Generates unique VCD variable names for ports even if those ports are connected to the same collapsed net. Optional.

- -vcdstim

  Ensures that port name order in the VCD file matches the declaration order in the instance's module or entity declaration. Optional. Refer to "Port Order Issues" for further information.

- <object_name> ...

  Specifies the Verilog or VHDL object or objects to add to the VCD file. Required. Multiple objects may be specified by separating names with spaces. Wildcards are accepted.

Examples

- Create a VCD file named *counter.vcd* of all IN ports in the region /test_design/dut/.

```
vcd dumpports -in -file counter.vcd /test_design/dut/*
```

- These two commands resimulate a design from a VCD file. Refer to "Simulating with Input Values from a VCD File" for further details.

```
vcd dumpports -file addern.vcd /testbench/uut/*
vsim -vcdstim addern.vcd addern -gn=8 -do "add wave /*; run 1000"
```

- This series of commands creates VCD files for the instances *proc* and *cache* and then re-simulates the design using the VCD files in place of the instance source files. Refer to "Replacing Instances with Output Values from a VCD File" for more information.

```
vcd dumpports -vcdstim -file proc.vcd /top/p/*
vcd dumpports -vcdstim -file cache.vcd /top/c/*
run 1000

vsim top -vcdstim /top/p=proc.vcd -vcdstim /top/c=cache.vcd
```

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpportsall

The **vcd dumpportsall** command creates a checkpoint in the VCD file which shows the value of all selected ports at that time in the simulation, regardless of whether the port values have changed since the last timestep.

Related Verilog task: $dumpportsall

Syntax

vcd dumpportsall [<filename>]

Arguments

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpportsflush

The **vcd dumpportsflush** command flushes the contents of the VCD file buffer to the specified VCD file.

Related Verilog task: $dumpportsflush

Syntax

vcd dumpportsflush [<filename>]

Arguments

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

vcd dumpportslimit

The **vcd dumpportslimit** command specifies the maximum size of the VCD file (by default, limited to available disk space). When the size of the file exceeds the limit, a comment is appended to the file and VCD dumping is disabled.

Related Verilog task: $dumpportslimit

Syntax

vcd dumpportslimit <dumplimit> [<filename>]

Arguments

- <dumplimit>

  Specifies the maximum VCD file size in bytes. Required.

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpportsoff

The **vcd dumpportsoff** command turns off VCD dumping and records all dumped port values as x.

Related Verilog task: $dumpportsoff

Syntax

vcd dumpportsoff [<filename>]

Arguments

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd dumpportson

The **vcd dumpportson** command turns on VCD dumping and records the current values of all selected ports. This command is typically used to resume dumping after invoking vcd dumpportsoff.

Related Verilog task: $dumpportson

Syntax

vcd dumpportson [<filename>]

Arguments

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on all open VCD files.

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd file

The **vcd file** command specifies the filename and state mapping for the VCD file created by a vcd add command. The **vcd file** command is optional. If used, it must be issued before any **vcd add** commands.

Related Verilog task: $dumpfile

Syntax

vcd file [-dumpports] [-direction] [<filename>] [-map <mapping pairs>] [-no_strength_range] [-nomap] [-unique]

Arguments

- -dumpports

  Capture detailed port driver data for Verilog ports and VHDL std_logic ports. Optional. This option works only on ports, and any subsequent vcd add command will accept only qualifying ports (silently ignoring all other specified objects).

- -direction

  Includes driver direction data in the VCD file. Optional.

- <filename>

  Specifies the name of the VCD file that is created (the default is *dump.vcd*). Optional.

- -map <mapping pairs>

  Affects only VHDL signals of type std_logic. Optional. It allows you to override the default mappings. The mapping is specified as a list of character pairs. The first character in a pair must be one of the std_logic characters UX01ZWLH- and the second character is the character you wish to be recorded in the VCD file. For example, to map L and H to z:

  ```
  vcd file -map "L z H z"
  ```

  Note that the quotes in the example above are a Tcl convention for command strings that include spaces.

- -no_strength_range

  Ignores strength ranges when resolving driver values. Optional. This argument is an extension to the IEEE 1364 specification. Refer to “Resolving Values” for additional information.

- -nomap

  Affects only VHDL signals of type std_logic. Optional. It specifies that the values recorded in the VCD file shall use the std_logic enumeration characters of UX01ZWLH-. This option results in a non-standard VCD file because VCD values are limited to the four state character set of x01z. By default, the std_logic characters are mapped as follows.

VHDL	VCD	VHDL	VCD
U	x	W	x
X	x	L	0
0	0	H	1
1	1	-	x
Z	z		

- -unique

  Generates unique VCD variable names for ports even if those ports are connected to the same collapsed net. Optional.

See also

"Value Change Dump (VCD) Files", vcd files, DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd files

The **vcd files** command specifies filenames and state mapping for VCD files created by the **vcd add** command. The **vcd files** command is optional. If used, it must be issued before any **vcd add** commands.

Related Verilog task: $fdumpfile

Syntax

vcd files [-compress] [-direction] <filename> [-map <mapping pairs>] [-no_strength_range] [-nomap] [-unique]

Arguments

- -compress

  Produces a compressed VCD file. Optional. ModelSim uses the gzip compression algorithm. If you specify a *.gz* extension on the **-file <filename>** argument, ModelSim compresses the file even if you don't use the **-compress** argument.

- -direction

  Includes driver direction data in the VCD file. Optional.

- <filename>

  Specifies the name of a VCD file to create. Required. 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>

  Affects only VHDL signals of type std_logic. Optional. It allows you to override the default mappings. The mapping is specified as a list of character pairs. The first character in a pair must be one of the std_logic characters UX01ZWLH- and the second character is the character you wish to be recorded in the VCD file. For example, to map L and H to z:

  ```
  vcd files -map "L z H z"
  ```

  Note that the quotes in the example above are a Tcl convention for command strings that include spaces.

- -no_strength_range

  Ignores strength ranges when resolving driver values. Optional. This argument is an extension to the IEEE 1364 specification. Refer to "Resolving Values" for additional information.

- -nomap

  Affects only VHDL signals of type std_logic. Optional. It specifies that the values recorded in the VCD file shall use the std_logic enumeration characters of UX01ZWLH-. This option

results in a non-standard VCD file because VCD values are limited to the four state character set of x01z. By default, the std_logic characters are mapped as follows.

VHDL	VCD	VHDL	VCD
U	x	W	x
X	x	L	0
0	0	H	1
1	1	-	x
Z	z		

- -unique

  Generates unique VCD variable names for ports even if those ports are connected to the same collapsed net. Optional.

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
```

See also

"Value Change Dump (VCD) Files", vcd file, DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd flush

The **vcd flush** command flushes the contents of the VCD file buffer to the specified VCD file. This command is useful if you want to create a complete VCD file without ending your current simulation.

Related Verilog tasks: $dumpflush, $fdumpflush

Syntax

vcd flush [<filename>]

Arguments

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd limit

The **vcd limit** command specifies the maximum size of a VCD file (by default, limited to available disk space).

When the size of the file exceeds the limit, a comment is appended to the file and VCD dumping is disabled.

Related Verilog tasks: $dumplimit, $fdumplimit

Syntax

vcd limit <filesize> [<filename>]

Arguments

- <filesize>

  (Required) Specifies the maximum VCD file size, in bytes. The numerical value of <filesize> can only be a whole number. You can use a unit multiplier of either Kb, Mb, or Gb with the numerical value (multipliers are case insensitive). Do not insert a space between the numerical value and the multiplier (for example, 400Mb, not 400 Mb).

- <filename>

  (Optional) Specifies the name of the VCD file. If omitted, the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

Example

- Specify a maximum VCD file size of 6 gigabytes and a VCD file named my_vcd_file.vcd.

```
vcd limit 6gb my_vcd_file.vcd
```

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd off

The **vcd off** command turns off VCD dumping to the specified file and records all VCD variable values as x.

Related Verilog tasks: $dumpoff, $fdumpoff

Syntax

vcd off [<filename>]

Arguments

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

See also

"Value Change Dump (VCD) Files"., DumpportsCollapse

Verilog tasks are documented in the IEEE 1364 standard.

vcd on

The **vcd on** command turns on VCD dumping to the specified file and records the current values of all VCD variables.

By default, **vcd on** is automatically performed at the end of the simulation time that the vcd add commands are performed.

Related Verilog tasks: $dumpon, $fdumpon

Syntax

vcd on [<filename>]

Arguments

- <filename>

  Specifies the name of the VCD file. Optional. If omitted the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

See also

"Value Change Dump (VCD) Files", DumpportsCollapse

Verilog system tasks are documented in the IEEE 1364 standard.

vcd2wlf

vcd2wlf is a utility that translates a VCD (Value Change Dump) file into a WLF file that you can display in ModelSim using the **vsim -view** argument. This command only works on VCD files containing positive time values.

Description

The **vcd2wlf** command functions as simple one-pass converter. If you are defining a bus in a VCD file, you must specify all bus bits before the next $scope or $upscope statement appears in the file. The best way to ensure that bits get converted together as a bus is to declare them on consecutive lines.

For example:

```
Line 21 : $var wire 1 $ in [2] $end
Line 22 : $var wire 1 $u in [1] $end
Line 23 : $var wire 1 # in [0] $end
```

Syntax

vcd2wlf [-splitio] [-splitio_in_ext <extension>] [-splitio_out_ext <extension>] [-nocase] <vcd filename> <wlf filename>

Arguments

- -splitio

  Specifies that extended VCD port values are to be split into their corresponding input and output components by creating 2 signals instead of just 1 in the resulting *.wlf* file. Optional. By default the new input-component signal keeps the same name as the original port name while the output-component name is the original name with "__o" appended to it.

- -splitio_in_ext <extension>

  Specifies an extension to add to input-component signal names created by using **-splitio**. Optional.

- -splitio_out_ext <extension>

  Specifies an extension to add to output-component signal names created by using **-splitio**. Optional.

- -nocase

  Converts all alphabetic identifiers to lowercase. Optional.

- <vcd filename>

  Specifies the name of the VCD file you want to translate into a WLF file. Required.

- <wlf filename>

  Specifies the name of the output WLF file. Required.

See also

"Value Change Dump (VCD) Files"

vcom

The **vcom** command compiles VHDL source code into a specified working library (or to the **work** library by default).

You can invoke **vcom** either from within ModelSim or from the command prompt of your operating system. You can invoke this command during simulation.

Compiled libraries are dependent on the major version of ModelSim. When moving between major versions, you must refresh compiled libraries using the **-refresh** argument to **vcom**. This is not required for minor versions (letter releases).

All arguments to the **vcom** command are case-sensitive. For example, -WORK and -work are not equivalent.

Syntax

vcom [-87] [-93] [-2002] [-amsstd | -noamsstd] [-bindAtCompile] [-bindAtLoad]
[-check_synthesis] [-debugVA] [-deferSubpgmCheck] [-explicit] [-f <filename>]
[-fsmnoresettrans] [-force_refresh <design_unit>] [-gen_xml <design_unit> <filename>]
[-help] [-ignoredefaultbinding] [-ignorevitalerrors] [-just abcep] [-line <number>] [-lint]
[-no1164] [-noaccel <package_name>] [-nocasestaticerror] [-nocheck]
[-nocoverrespecthandl] [-nodbgsym] [-noDeferSubpgmCheck] [-noindexcheck] [-nologo]
[-nonstddriverinit] [-noothersstaticerror] [-norangecheck]
[-note <msg_number> [,<msg_number>, …]] [-novital] [-novitalcheck]
[-nowarn <category_number>] [-O0]
[-pedanticerrors] [-performdefaultbinding] [-quiet] [-rangecheck] [-refresh]
[-s] [-skip abcep] [-source]
[-time] [-togglecountlimit] [-togglewidthlimit] [-version]
[-suppress <msg_number>[,<msg_number>,…]]
[-error <msg_number>[,<msg_number>,…]]
[-warning <msg_number>[,<msg_number>,…]]
[-fatal <msg_number>[,<msg_number>,…]]
[-work <library_name>] <filename>

Arguments

- -87

  Disables support for VHDL-1993 and 2002. Optional. Default is -2002. See additional discussion in the examples. You can modify the VHDL93 variable in the *modelsim.ini* file to set this permanently (Refer to “Simulator Control Variables”).

- -93

  Disables support for VHDL-1987 and 2002. Optional. Default is -2002. See additional discussion in the examples. You can modify the VHDL93 variable in the *modelsim.ini* file to set this permanently.

- -2002

  Specifies that the compiler is to support VHDL-2002. Optional. This is the default.

- -allowProtectedBeforeBody

  Allows a variable of a protected type to be created prior to declaring the body. Optional.

- -amsstd | -noamsstd

  Specifies whether vcom adds the declaration of REAL_VECTOR to the STANDARD package. This is useful for designers using VHDL-AMS to test digital parts of their model.

  -amsstd — REAL_VECTOR is included in STANDARD.

  -noamsstd — REAL_VECTOR is not included in STANDARD (default).

  You can also control this with the AmsStandard variable or the MGC_AMS_HOME environment variable.

- -bindAtCompile

  Forces ModelSim to perform default binding at compile time rather than at load time. Optional. Refer to "Default Binding" for more information. You can change the permanent default by editing the BindAtCompile variable in the *modelsim.ini*.

- -bindAtLoad

  Forces ModelSim to perform default binding at load time rather than at compile time. Optional. Default.

- -check_synthesis

  Turns on limited synthesis rule compliance checking. Specifically, it checks to see that signals read by a process are in the sensitivity list. Optional. The checks understand only combinational logic, not clocked logic. Edit the CheckSynthesis variable in the *modelsim.ini* file to set a permanent default.

- -debugVA

  Prints a confirmation if a VITAL cell was optimized, or an explanation of why it was not, during VITAL level-1 acceleration. Optional.

- -deferSubpgmCheck

  Forces the compiler to report array indexing and length errors as warnings (instead of as errors) when encountered within subprograms. Subprograms with indexing and length errors that are invoked during simulation cause the simulator to report errors, which can potentially slow down simulation because of additional checking.

- -error <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "error." Optional. Edit the error variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -explicit

  Directs the compiler to resolve ambiguous function overloading by favoring the explicit function definition over the implicit function definition. Optional. Strictly speaking, this behavior does not match the VHDL standard. However, the majority of EDA tools choose explicit operators over implicit operators. Using this switch makes ModelSim compatible with common industry practice.

- -f <filename>

  Specifies a file with more command-line arguments. Optional. Allows complex argument strings to be reused without retyping. Allows gzipped input files. Nesting of **-f** options is allowed.

  Refer to the section "Argument Files" for more information.

- -fsmnoresettrans

  Disables recognition of implicit asynchronous reset transitions. Optional. This has the effect of excluding asynchronous reset transitions from any coverage results.

- -fatal <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "fatal." Optional. Edit the fatal variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -force_refresh <design_unit>

  Forces the refresh of all specified design units. Optional. By default, the work library is updated; use **-work <library_name>**, in conjunction with **-force_refresh**, to update a different library (for example, vcom -work <your_lib_name> -force_refresh).

  When the compiler refreshes a design unit, it checks each dependency to ensure its source has not been changed and recompiled. Sometimes the tool's dependency checking algorithm changes from release to release. This can lead to false errors during the integrity checks performed by the **-refresh** argument. An example of such a message follows:

  ```
  ** Error: (vsim-13) Recompile /u/test/dware/dware_61e_beta.dwpackages
  because /home/users/questasim/linux/../synopsys.attributes has changed.
  ```

  The **-force_refresh** argument forces the refresh of the design unit, overriding any dependency checking errors encountered by the **-refresh** argument.

  A more conservative approach to working around **-refresh** dependency checks is to recompile the source code, if it is available.

- -gen_xml <design_unit> <filename>

  Produces an XML-tagged file containing the interface definition of the specified entity. Optional. This option requires a two-step process where you must 1) compile <filename> into a library with **vcom** (without **-gen_xml**) then 2) execute **vcom** with the **-gen_xml** switch, for example:

  ```
  vlib work
  vcom counter.vhd
  ```

```
vcom -gen_xml counter counter.xml
```

- -help

  Displays the command's options and arguments. Optional.

- -ignoredefaultbinding

  Instructs the compiler not to generate a default binding during compilation. Optional. You must explicitly bind all components in the design to use this switch.

- -ignorevitalerrors

  Directs the compiler to ignore VITAL compliance errors. Optional. The compiler still reports that VITAL errors exist, but it will not stop the compilation. You should exercise caution in using this switch; as part of accelerating VITAL packages, we assume that compliance checking has passed.

- -just abcep

  Directs the compiler to "just" include:

  a — architectures

  b — bodies

  c — configurations

  e — entities

  p — packages

  Any combination in any order can be used, but one choice is required if you use this optional switch.

- -line <number>

  Starts the compiler on the specified line in the VHDL source file. Optional. By default, the compiler starts at the beginning of the file.

- -lint

  Optional. Enables better checking on case statement rules. Also enables warning messages for the following situations: 1) the result of the built-in concatenation operator ("&") is the actual for a subprogram formal parameter of an unconstrained array type; 2) 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; 3) a direct recursive subprogram call; and 4) in cases involving class SIGNAL formal parameters, as described in IEEE Standard VHDL Language Reference Manual 1076-1993, section 2.1.1.2 entitled "Signal parameters", line 115. This last check only applies to designs compiled using 87. If you were to compile in 93, it would be flagged as a warning or error, even without the -lint argument. Can also be enabled using the Show_Lint variable in the *modelsim.ini* file.

- -no1164

  Causes the source files to be compiled without taking advantage of the built-in version of the IEEE **std_logic_1164** package. Optional. This will typically result in longer simulation times for VHDL programs that use variables and signals of type **std_logic**.

- -noaccel <package_name>

  Turns off acceleration of the specified package in the source code using that package.

- -nocasestaticerror

  Suppresses case statement static warnings. Optional. VHDL standards require that case statement alternative choices be static at compile time. However, some expressions which are globally static are allowed. This switch prevents the compiler from warning on such expressions. If the **-pedanticerrors** switch is specified, this switch is ignored.

- -nocheck

  Disables index and range checks. Optional. You can disable these individually using the **-noindexcheck** and **-norangecheck** arguments, respectively.

- -nocoverrespecthandl

  Specifies that you want the VHDL 'H' and 'L' input values on conditions and expressions to be automatically converted to ‘1’ and ‘0’, respectively. By default in the current release, they are not automatically converted.

  As an alternative to using this argument — if you are not using 'H' and 'L' values and don’t want the additional UDP rows that are difficult to cover — you can either:

  - change your VHDL expressions of the form (a = '1') to (to_x01(a) = '1') or to std_match(a,'1'). These functions are recognized and serve to simplify the UDP tables
  - set the CoverRespectHandL .ini file variable to 0

- -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 allowing you to view detailed information about design objects at the source level. Two major GUI features that use this database include source window annotation and textual dataflow.

  You should only specify this switch if you know that anyone using the library will not require this information for design analysis purposes.

- -noDeferSubpgmCheck

  Causes range and length violations detected within subprograms to be reported as errors (instead of as warnings). As an alternative to using this argument, you can set the NoDeferSubpgmCheck property in the modelsim.ini file to a value of 1.

- -noindexcheck

  Disables checking on indexing expressions to determine whether indices are within declared array bounds. Optional.

- -nologo

  Disables display of the startup banner. Optional.

- -nonstddriverinit

  Forces ModelSim to match pre-5.7c behavior in initializing drivers in a particular case. Optional. Prior to 5.7c, VHDL ports of mode out or inout could have incorrectly initialized drivers if the port did not have an explicit initialization value and the actual signal connected to the port had explicit initial values. Depending on a number of factors, ModelSim could incorrectly use the actual signal's initial value when initializing lower level drivers. Note that the argument does not cause all lower-level drivers to use the actual signal's initial value. It does this only in the specific cases where older versions used the actual signal's initial value.

- -noothersstaticerror

  Disables warnings that result from array aggregates with multiple choices having "others" clauses that are not locally static. Optional. If the **-pedanticerrors** switch is specified, this switch is ignored.

- -norangecheck

  Disables run time range checking. In some designs, this results in a 2X speed increase. Range checking is enabled by default or, once disabled, can be enabled using **-rangecheck**. Refer to "Range and Index Checking" for additional information.

- -note <msg_number> [,<msg_number>, …]

  Changes the severity level of the specified message(s) to "note." Optional. Edit the note variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -novital

  Causes **vcom** to use VHDL code for VITAL procedures rather than the accelerated and optimized timing and primitive packages built into the simulator kernel. Optional. Allows breakpoints to be set in the VITAL behavior process and permits single stepping through the VITAL procedures to debug your model. Also all of the VITAL data can be viewed in the Locals or Objects windows.

- -novitalcheck

  Disables Vital level 1 checks and also Vital level 0 checks defined in section 4 of the Vital-95 Spec (IEEE Std 1076.4-1995). Optional.

- -nowarn <category_number>

  Selectively disables a category of warning messages. Optional. Multiple **-nowarn** switches are allowed. Warnings may be disabled for all compiles via the Main window **Compile >**

Compile Options menu command or the *modelsim.ini* file (Refer to "VHDL Compiler Control Variables").

The warning message categories are described in Table 2-4:

Table 2-4. 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
13	constructs that coverage can't handle
14	locally static error deferred until simulation run

- -O0

  Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out. Add the DisableOpt variable to the [vcom] section of the *modelsim.ini* file to set a permanent default.

- -pedanticerrors

  Forces ModelSim to error (rather than warn) on a variety of conditions. Refer to "Enforcing Strict 1076 Compliance" for a complete list. Optional. This argument overrides **-nocasestaticerror** and **-noothersstaticerror** (see above).

- -performdefaultbinding

  Enables default binding when it has been disabled via the **RequireConfigForAllDefaultBinding** option in the *modelsim.ini* file. Optional.

- -quiet

  Disables 'Loading' messages. Optional.

- -rangecheck

  Enables run time range checking. Default. Range checking can be disabled using the **-norangecheck** argument. Refer to "Range and Index Checking" for additional information.

- -refresh

  Regenerates a library image. Optional. By default, the work library is updated. To update a different library, use **-work <library_name>** with **-refresh** (for example, vcom -work <your_lib_name> -refresh). If a dependency checking error occurs which prevents the refresh, use the **vcom -force_refresh** argument. See the **vcom** Examples for more information. You may use a specific design name with **-refresh** to regenerate a library image for that design, but you may not use a file name.

- -s

  Instructs the compiler not to load the **standard** package. Optional. This argument should only be used if you are compiling the **standard** package itself.

- -skip abcep

  Directs the compiler to skip all:

  a — architectures

  b — bodies

  c — configurations

  e — entities

  p — packages

  Any combination in any order can be used, but one choice is required if you use this optional switch.

- -source

  Displays the associated line of source code before each error message that is generated during compilation. Optional. By default, only the error message is displayed.

- -suppress <msg_number>[,<msg_number>,…]

  Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress Fatal or Internal messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -time

  Reports the "wall clock time" **vcom** takes to compile the design. Optional. Note that if many processes are running on the same system, wall clock time may differ greatly from the actual "cpu time" spent on **vcom**.

- -togglecountlimit

  Limits the toggle coverage count for a toggle node. Optional. After the limit is reached, further activity on the node will be ignored for toggle coverage. All possible transition edges

must reach this count for the limit to take effect. For example, if you are collecting toggle data on 0->1 and 1->0 transitions, both transition counts must reach the limit. If you are collecting "full" data on 6 edge transitions, all 6 must reach the limit. Overrides the global value set by the ToggleCountLimit *modelsim.ini* variable.

- -togglewidthlimit

  Sets the maximum width of signals that are automatically added to toggle coverage with the **-cover t** argument. Optional. Can be set on design unit basis. Overrides the global value set by the ToggleWidthLimit *modelsim.ini* variable.

- -version

  Returns the version of the compiler as used by the licensing tools. Optional.

- -warning <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "warning." Optional. Edit the warning variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -work <library_name>

  Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional; by default, the compiled design units are added to the **work** library. The specified pathname overrides the pathname specified for work in the project file.

- <filename>

  Specifies the name of a file containing the VHDL source to be compiled. One filename is required; multiple filenames can be entered separated by spaces or wildcards may be used (e.g., **.vhd*).

  If you don't specify a filename, and you are using the GUI, a dialog box pops up allowing you to select the options and enter a filename.

Examples

- Compile the VHDL source code contained in the file *example.vhd*.

  ```
  vcom example.vhd
  ```

- ModelSim supports designs that use elements conforming to the 1987, 1993, and 2002 standards. Compile the design units separately using the appropriate switches.

  ```
  vcom -87 o_units1.vhd o_units2.vhd
  vcom -93 n_unit91.vhd n_unit92.vhd
  ```

- When compiling source that uses the **numeric_std** package, this command turns off acceleration of the **numeric_std** package, located in the **ieee** library.

  ```
  vcom -noaccel numeric_std example.vhd
  ```

- Although it is not obvious, the = operator is overloaded in the **std_logic_1164** package. All enumeration data types in VHDL get an "implicit" definition for the = operator. So while there is no explicit = operator, there is an implicit one. This implicit declaration

can be hidden by an explicit declaration of = in the same package (LRM Section 10.3). However, if another version of the = operator is declared in a different package than that containing the enumeration declaration, and both operators become visible through **use** clauses, neither can be used without explicit naming.

```
vcom -explicit example.vhd
```

To eliminate that inconvenience, the VCOM command has the **-explicit** option that allows the explicit = operator to hide the implicit one. Allowing the explicit declaration to hide the implicit declaration is what most VHDL users expect.

```
ARITHMETIC."="(left, right)
```

- The **-work** option specifies **mylib** as the library to regenerate. **-refresh** rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of ModelSim (4.6 and later only).

```
vcom -work mylib -refresh
```

a test-associated merge, which associates coverage items with the test(s) that covered them. To obtain a more basic level of information use the argument without **-test.**

Important: When the **-metric aggregate** argument is used, the resulting metric number will not “match” any other total coverage number produced by other verification tools (i.e. coverage analyze). This is important because when you use any of the arguments (**-totals**, **-goal**, with ranktest command, the aggregate metric is the default.

vdel

The **vdel** command deletes a design unit from a specified library.

Syntax

vdel [-help] [-lib <library_name>] [-verbose] [-all | <design_unit> [<arch_name>]

Arguments

- -all

  Deletes an entire library. Optional. BE CAREFUL! Libraries cannot be recovered once deleted, and you are not prompted for confirmation.

- <arch_name>

  Specifies the name of an architecture to be deleted. Optional. If omitted, all of the architectures for the specified entity are deleted. Invalid for a configuration or a package.

- <design_unit>

  Specifies the entity, package, configuration, or module to be deleted. Required unless **-all** is used.

- -help

  Displays the command's options and arguments. Optional.

- -lib <library_name>

  Specifies the logical name or pathname of the library that holds the design unit to be deleted. Optional. By default, the design unit is deleted from the **work** library.

- -verbose

  Displays progress messages. Optional.

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

The **vdir** command lists the contents of a design library.

This command can also be used to check compatibility of a vendor library. If **vdir** cannot read a vendor-supplied library, the library may not be ModelSim compatible.

Syntax

vdir [-help] [-l | [-prop <prop>]] [-r] [-all | [-lib <library_name>]] [<design_unit>]

Arguments

- -help

  Displays options and arguments for this command. Optional.

- -l

  Prints the version of **vcom** or **vlog** with which each design unit was compiled, plus any compilation options used. Also prints the object-code version number that indicates which versions of **vcom/vlog** and ModelSim are compatible.

- -prop <prop>

  Reports on the specified design unit property, as listed in Table 2-5. If you do not specify a value for <prop>, an error message is displayed.

Table 2-5. Design Unit Properties

Value of <prop>	Description
archcfg	configuration for arch
bbox	blackbox for optimized design
body	needs a body
default	default options
dir	source directory
dpnd	depends on
entcfg	configuration for entity
extern	package reference number
inline	module inlined
lrm	language standard
mtime	source modified time
name	short name
opcode	opcode format
options	compile options

Table 2-5. Design Unit Properties

Value of <prop>	Description
root	optimized Verilog design root
src	source file
top	top level model
ver	version number
vlogv	Verilog version
voptv	Verilog optimized version

- -r

  Prints architecture information for each entity in the output.

- -all

  Lists the contents of all libraries listed in the Library section of the active *modelsim.ini* file. Optional. Refer to "Library Path Variables" for more information.

- -lib <library_name>

  Specifies the logical name or the pathname of a library to be listed. Optional. By default, the contents of the **work** library are listed.

- <design_unit>

  Indicates the design unit to search for within the specified library. If the design unit is a VHDL entity, its architectures are listed. Optional. By default all entities, configurations, modules, packages, and optimized design units in the specified library are listed.

Examples

- List the architectures associated with the entity named **my_asic** that reside in the HDL design library called **design**.

  ```
  vdir -lib design my_asic
  ```

- Show the output of **vdir -l,** including any compilation options used to compile the library:

  ```
  > # MODULE ram_tb
  > #     Verilog Version: RV9i]?9FGhibjG<jXXV_`1
  > #     Version number: CRW2<UhheaW;LIL2_B5o31
  > #     Source modified time: 1132284874
  > #     Source file: ram_tb.v
  > #     Version number: CRW2<UhheaW;LIL2_B5o31
  > #     Opcode format: 6.1c; VLOG SE Object version 31
  > #     Optimized Verilog design root: 1
  > #     Language standard: 1
  > #     Compile options: -cover bcst
  > #     Compile defaults: GenerateLoopIterationMax=100000
  > #     Source directory: C:\Verif\QuestaSim_6.1c
                              \examples\tutorials\verilog\memory
  ```

vencrypt

The **vencrypt** command encrypts Verilog and SystemVerilog code contained within encryption envelopes. The code is not pre-processed before encryption, so macros and other `directives are unchanged. This allows IP vendors to deliver encrypted IP with undefined macros and `directives.

Upon execution of this command, the filename extension will be changed to .vp for Verilog files (.v files) and .svp for SystemVerilog files (.sv files). As the vencrypt utility processes the file (or files), if it does not find any encryption directives it reprocesses the file using the following default encryption:

```
`pragma protect data_method = "aes128-cbc"
`pragma protect key_keyowner = "MTI"
‘pragma protect key_keyname = "MGC-DVT-MTI"
‘pragma protect key_method = "rsa"
`pragma protect key_block encoding = (enctype = "base64")
`pragma protect begin
```

The vencrypt command must be followed by a compile command – such as vlog – for the design to be compiled.

Syntax

vencrypt <filename> [-d <dirname>] [-e <extension>] [-f <filename>] [-h <filename>] [-help] [-l <filename>] [-o <filename>] [-p <prefix>] [-quiet]

- <filename>

  Specifies the name of the Verilog source code file to encrypt. One filename is required. Multiple filenames can be entered separated by spaces. Wildcards can be used. Default encryption pragmas will be used, as described above, if no encryption directives are found during processing.

- -d <dirname>

  Specifies directory that will contain encrypted Verilog files. Optional. If no directory is specified, current working directory will be used. The original file extension (*.v* for Verilog and *.sv* for SystemVerilog) will be preserved.

- -e <extension>

  Specifies a filename extension. Optional.

- -f <filename>

  Specifies a file with more command line arguments. Optional. Allows complex arguments to be reused without retyping. Nesting of **-f** options is allowed.

  Refer to the section "Argument Files" for more information.

- -h <filename>

  Concatenates header information, specified by <file>, into all design files listed with <filename>. Optional. Allows the user to pass a large number of files to the vencrypt utility that do not contain the **`pragma protect** or **`protect** information about how to encrypt the

file. Saves the user from editing hundreds of files to add in the same `pragma protect to every file.

- -help

  Displays vencrypt command arguments. Optional.

- -l <filename>

  Redirects output to the file designated by <filename>. Optional.

- -o <filename>

  Combines all encrypted output into a single file. Optional.

- -p <prefix>

  Prepends file names with a prefix. Optional.

- -quiet

  Disables encryption messages. Optional.

See also

"Protecting Your Source Code" in the User's Manual

Example

- Insert header information into all design files listed.

  ```
  vencrypt -h encrypt_head top.v cache.v gates.v memory.v
  ```

  The *encrypt_head* file may look like the following:

  ```
  `pragma protect data_method = "aes128-cbc"
  `pragma protect author = "IP Provider"
  `pragma protect key_keyowner = "MTI", key_method = "rsa"
  `pragma protect key_keyname = "MGC-DVT-MTI"
  `pragma protect begin
  ```

  There is no **`pragma protect end** expression in the header file, just the header block that starts the encryption. The **`pragma protect end** expression is implied by the end of the file. For more detailed examples, see "Protecting Your Source Code" in the User's Manual.

verror

The **verror** command prints a detailed description about a message number. It may also point to additional documentation related to the error.

Syntax

verror [-fmt | -tag | -fmt -tag | -full] <msgNum> ...

verror [-fmt | -tag | -fmt -tag | -full] [-tool <tool>] -all

verror -ranges

verror -help

Arguments

- -fmt | -tag | -full

  Specifies the type and amount of information to return.

  -fmt — returns the format string used in the error message.

  -tag — returns a tag associated with the error message.

  -full — returns the format string, tag, and complete text associated with the error message.

- [-tool <tool>] -all

  Allows you to return information about all error messages.

  -all — returns all error messages.

  -tool <tool> -all — returns all error messages associated with the specified tool, where <tool> can be one of the following:

common	vcom	vcom-vlog
vlog	vsim	vsim-vish
wlf	vsim-sccom	sccom
vsim-systemc	ucdb	vsim-vlog
pseudo_synth		

- <msgNum>

  Specifies the message number(s) you would like more information about. You can find the message number in messages of the format:

  ** <Level>: ([<Tool>-[<Group>-]]<MsgNum>) <FormattedMsg>

  You can specify <msgNum> any number of times for one verror command. It is suggested that you use a space-separated list.

- -ranges

  Prints the numeric ranges of error message numbers, organized by tool.

Example

- If you receive the following message in the transcript:

  ```
  ** Error (vsim-3061) foo.v(22): Too many Verilog port connections.
  ```

  and you would like more information about this message, you would type:

  ```
  verror 3061
  ```

  and receive the following output:

  ```
  Message # 3061:
  Too many Verilog ports were specified in a mixed VHDL/Verilog
  instantiation. Verify that the correct VHDL/Verilog connection is
  being made and that the number of ports matches.
  [DOC: ModelSim User's Manual - Mixed VHDL and Verilog Designs
  Chapter]
  ```

vgencomp

Once a Verilog module is compiled into a library, you can use the **vgencomp** command to write its equivalent VHDL component declaration to standard output.

Optional switches allow you to generate bit or vl_logic port types; std_logic port types are generated by default.

Syntax

vgencomp [-help] [-lib <library_name>] [-b] [-s] [-v] <module_name>

Arguments

- -help

  Displays the command's options and arguments. Optional.

- -lib <library_name>

  Specifies the pathname of the working library. If not specified, the default library **work** is used. Optional.

- -b

  Causes **vgencomp** to generate bit port types. Optional.

- -s

  Used for the explicit declaration of default std_logic port types. Optional.

- -v

  Causes **vgencomp** to generate vl_logic port types. Optional.

- <module_name>

  Specifies the name of the Verilog module to be accessed. Required.

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;
```

view

The **view** command displays a stand-alone window or Main window pane.

To remove a window, use the noview command.

The **view** command without arguments returns a list of window class names of all the windows currently open.

The **view** command with one or more options and no window classes or window names specified applies the options to the currently open windows. See examples for additional details.

Syntax

view [*] [-height <n>] [-icon] [-title {New Window Title}] [-undock | -dock]
[-width <n>] [-x <n>] [-y <n>] <window_type>…

Arguments

- *

  Specifies that all windows be opened. Optional.

- -height <n>

  Specifies the window height in pixels. Valid only for stand-alone windows, not panes in the Main window. Optional.

- -icon

  Toggles the view between window and icon. Valid only for stand-alone windows, not panes in the Main window. Optional.

- -title {New Window Title}

  Specifies the window title of the designated window. Curly braces are only needed for titles that include spaces. Double quotes can be used in place of braces, for example "New Window Title". If the new window title does not include spaces, no braces or quotes are needed. For example: *-title new_wave wave* assigns the title *new_wave* to the Wave window.

- -undock

  Opens the specified pane as a standalone window, undocked from the Main window. Optional.

- -dock

  Docks the specified standalone window into the Main window.

- -width <n>

  Specifies the window width in pixels. Valid only for stand-alone windows, not panes in the Main window. Optional.

- <window_type>…

  Specifies the window/pane type to view. Required. You do not need to type the full type name (see examples below); implicit wildcards are accepted; multiple window types are accepted. Available window/pane types are:

  capacity,dataflow, list, locals, memory, objects, process, profiledetails, profilemain, signals, structure, variables, wave, watch, workspace

- -x <n>

  Specifies the window upper-left-hand x-coordinate in pixels. Valid only for stand-alone windows, not panes in the Main window. Optional.

- -y <n>

  Specifies the window upper-left-hand y-coordinate in pixels. Valid only for stand-alone windows, not panes in the Main window. Optional.

Examples

- Undock the Wave pane from the Main window and makes it a standalone window.

  ```
  view -undock wave
  ```

- Undock all currently open panes in the Main window.

  ```
  view -undock
  ```

- Display the Watch and Wave panes.

  ```
  view w
  ```

- Display the Objects and Active Process panes.

  ```
  view ob pr
  ```

- Open a new Wave window with My Wave Window as its title.

  ```
  view -title {My Wave Window} wave
  ```

See also

noview

virtual count

The **virtual count** command counts the number of currently defined virtuals that were not read in using a macro file.

Syntax

virtual count [-kind <kind>] [-unsaved]

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

- -unsaved

  Specifies that the count include only those virtuals that have not been saved to a macro file. Optional.

See also

virtual define, virtual save, virtual show, "Virtual Objects"

virtual define

The **virtual define** command prints to the Transcript pane the definition of the virtual signal or function in the form of a command that can be used to re-create the object.

Syntax

virtual define [-kind <kind>] <pathname>

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

- <pathname>

  Specifies the path to the virtual(s) for which you want definitions. Required. Wildcards can be used.

Examples

- Show the definitions of all the virtuals you have explicitly created.

```
virtual define -kind explicits *
```

See also

virtual describe, virtual show, "Virtual Objects"

virtual delete

The **virtual delete** command removes the matching virtuals.

Syntax

virtual delete [-kind <kind>] <pathname>

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

- <pathname>

  Specifies the path to the virtual(s) you want to delete. Required. Wildcards can be used.

Examples

- Delete all of the virtuals you have explicitly created.

```
virtual delete -kind explicits *
```

See also

virtual signal, virtual function, "Virtual Objects"

virtual describe

The **virtual describe** command prints to the Transcript pane a complete description of the data type of one or more virtual signals.

Similar to the existing **describe** command.

Syntax

virtual describe [-kind <kind>] <pathname>

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

- <pathname>

  Specifies the path to the virtual(s) for which you want descriptions. Required. Wildcards can be used.

Examples

- Describe the data type of all virtuals you have explicitly created.

```
virtual describe -kind explicits *
```

See also

virtual define, virtual show, "Virtual Objects"

virtual expand

The **virtual expand** command produces a list of all the non-virtual objects contained in the specified virtual signal(s).

This can be used to create a list of arguments for a command that does not accept or understand virtual signals.

Syntax

virtual expand [-base] <pathname>

Arguments

- -base

  Causes the root signal parent to be output in place of a subelement. Optional. For example:

  ```
  vcd add [virtual expand -base myVirtualSignal]
  ```

  the resulting command after substitution would be:

  ```
  vcd add signala signalb signalc
  ```

- <pathname>

  Specifies the path to the signals and virtual signals to expand. Required. Wildcards can be used. Any number of paths can be specified.

Examples

- Add the elements of a virtual signal to the VCD file.

  In the Tcl language, the square brackets specify that the enclosed command should be executed first ("virtual expand …"), then the result substituted into the surrounding command.

  ```
  vcd add [virtual expand myVirtualSignal]
  ```

  So if myVirtualSignal is a concatenation of signala, signalb.rec1 and signalc(5 downto 3), the resulting command after substitution would be:

  ```
  vcd add signala signalb.rec1 {signalc(5 downto 3)}
  ```

  The slice of signalc is quoted in curly braces, because it contains spaces.

See also

virtual signal, “Virtual Objects”

virtual function

The **virtual function** command creates a new signal, known only by the GUI (not the kernel), that consists of logical operations on existing signals and simulation time, as described in **<expressionString>**.

It cannot handle bit selects and slices of Verilog registers. Please see *Syntax and Conventions* for more details on syntax.

If the virtual function references more than a single scalar signal, it will display as an expandable object in the Wave and Objects windows. The children correspond to the inputs of the virtual function. This allows the function to be "expanded" in the Wave window to see the values of each of the input waveforms, which could be useful when using virtual functions to compare two signal values.

Virtual functions can also be used to gate the List window display.

Syntax

virtual function [-env <path>] [-install <path>] [-delay <time>] {<expressionString>} <name>

Arguments

Arguments for **virtual function** are the same as those for **virtual signal**, except for the contents of the expression string.

- -env <path>

  Specifies a hierarchical context for the signal names in **<expressionString>** so they don't all have to be full paths. Optional.

- -install <path>

  Causes the newly-created signal to become a child of the specified region. If **-install** is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in **<expressionString>**. If the expression references more than one WLF file (dataset), the virtual signal will automatically be placed in region virtuals:/Functions. Optional.

- -delay <time>

  Specifies a value by which the virtual function will be delayed. Optional. You can use negative values to look forward in time. If units are specified, the <time> option must be enclosed in curly braces. See the examples below for more details.

- {<expressionString>}

  A text string expression in the MTI GUI expression format. Required. See GUI_expression_format for more information.

- <name>

  The name you define for the virtual signal. Required. Case is ignored unless installed in a Verilog region. Use alpha, numeric, and underscore characters only, unless you are using

VHDL extended identifier notation. If using VHDL extended identifier notation, **<name>** needs to be quoted with double quotes or with curly braces.

Examples

- Create a signal */chip/section1/clk_n* that is the inverse of */chip/section1/clk*.

```
virtual function { not /chip/section1/clk } clk_n
```

- Create a std_logic_vector equivalent of a Verilog register *rega* and installs it as */chip/rega_slv*.

```
virtual function -install /chip { (std_logic_vector) chip.vlog.rega
} rega_slv
```

- Create a boolean signal */chip/addr_eq_fab* that is true when */chip/addr[11:0]* is equal to hex "fab", and false otherwise. It is acceptable to mix VHDL signal path notation with Verilog part-select notation.

```
virtual function { /chip/addr[11:0] == 0xfab } addr_eq_fab
```

- Create a signal that is high only during times when signal */chip/siga* of the gate-level version of the design does not match */chip/siga* of the rtl version of the design. Because there is no common design region for the inputs to the expression, *siga_diff* is installed in region *virtuals:/Functions*. The virtual function *siga_diff* can be added to the Wave window, and when expanded will show the two original signals that are being compared.

```
virtual function { gate:/chip/siga XOR rtl:/chip/siga } siga_diff
```

- Create a virtual signal consisting of the logical "AND" function of */top/signalA* with */top/signalB*, and delays it by 10 ns.

```
virtual function -delay {10 ns} {/top/signalA AND /top/signalB}
myDelayAandB
```

- Create a one-bit signal *outbus_diff* which is non-zero during times when any bit of */chip/outbus* in the gate-level version doesn't match the corresponding bit in the rtl version.

  This expression uses the "OR-reduction" operator, which takes the logical OR of all the bits of the vector argument.

```
virtual function { | (gate:/chip/outbus XOR rtl:/chip/outbus) }
outbus_diff
```

Commands fully compatible with virtual functions

add log	delete	describe
log		("virtual describe" is a little faster)

examine find restart
searchlog show

Commands not currently compatible with virtual functions

drivers force noforce
vcd add when

See also

virtual count virtual define virtual delete
virtual describe virtual expand virtual hide
virtual log virtual nohide virtual nolog
virtual region virtual save virtual show
virtual signal virtual type "Virtual Objects

virtual hide

The **virtual hide** command causes the specified real or virtual signals to not be displayed in the Objects window. This is used when you want to replace an expanded bus with a user-defined bus.

You make the signals reappear using the **virtual nohide** command.

Syntax

virtual hide [-kind <kind>]|[-region <path>] <pattern>

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

- -region <path>

  Used in place of -kind to specify a region of design space in which to look for the signal names. Optional.

- <pattern>

  Indicates which signal names or wildcard patterns should be used in finding the signals to hide. Required. Any number of names or wildcard patterns may be used.

See also

virtual nohide, "Virtual Objects"

virtual log

The **virtual log** command causes the simulation-mode dependent signals of the specified virtual signals to be logged by the kernel.

If wildcard patterns are used, it will also log any normal signals found, unless the **-only** option is used. You unlog the signals using the **virtual nolog** command.

Syntax

virtual log [-kind <kind>] | [-region <path>] [-recursive] [-only] [-in] [-out] [-inout] [-internal] [-ports] <pattern>

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

- -region <path>

  Used in place of **-kind** to specify a region of design space in which to look for signals to log. Optional.

- -recursive

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

- -only

  Can be used with a wildcard to specify that only virtual signals (as opposed to all signals) found by the wildcard should be logged. Optional.

- -in

  Specifies that the kernel log data for ports of mode IN whose names match the specification. Optional.

- -out

  Specifies that the kernel log data for ports of mode OUT whose names match the specification. Optional.

- -inout

  Specifies that the kernel log data for ports of mode INOUT whose names match the specification. Optional.

- -internal

  Specifies that the kernel log data for internal (non-port) objects whose names match the specification. Optional.

- -ports

  Specifies that the kernel log data for all ports. Optional.

- <pattern>

  Indicates which signal names or wildcard patterns should be used in finding the signals to log. Required. Any number of names or wildcard patterns may be used.

See also

virtual nolog, “Virtual Objects”

virtual nohide

The **virtual nohide** command reverses the effect of a **virtual hide** command, causing the specified real or virtual signals to reappear the Objects window.

Syntax

virtual nohide [-kind <kind>]|[-region <path>] <pattern>

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

- -region <path>

  Used in place of **-kind** to specify a region of design space in which to look for the signal names. Optional.

- <pattern>

  Indicates which signal names or wildcard patterns should be used in finding the signals to expose. Required. Any number of names or wildcard patterns may be used.

See also

virtual hide, "Virtual Objects"

virtual nolog

The **virtual nolog** command reverses the effect of a **virtual log** command. It causes the simulation-dependent signals of the specified virtual signals to be excluded ("unlogged") by the kernel.

If wildcard patterns are used, it will also unlog any normal signals found, unless the **-only** option is used.

Syntax

virtual nolog [-kind <kind>] | [-region <path>] [-recursive] [-only] [-in] [-out] [-inout] [-internal] [-ports] <pattern>

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

- -region <path>

  Used in place of **-kind** to specify a region of design space in which to look for signals to unlog. Optional.

- -recursive

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

- -only

  Can be used with a wildcard to specify that only virtual signals (as opposed to all signals) found by the wildcard should be unlogged. Optional.

- -in

  Specifies that the kernel exclude data for ports of mode IN whose names match the specification. Optional.

- -out

  Specifies that the kernel exclude data for ports of mode OUT whose names match the specification. Optional.

- -inout

  Specifies that the kernel exclude data for ports of mode INOUT whose names match the specification. Optional.

- -internal

  Specifies that the kernel exclude data for internal (non-port) objects whose names match the specification. Optional.

- -ports

  Specifies that the kernel exclude data for all ports. Optional.

- <pattern>

  Indicates which signal names or wildcard pattern should be used in finding the signals to unlog. Required. Any number of names or wildcard patterns may be used.

See also

virtual log, “Virtual Objects”

virtual region

The **virtual region** command creates a new user-defined design hierarchy region.

Syntax

virtual region <parentPath> <regionName>

Arguments

- <parentPath>

  The full path to the region that will become the parent of the new region. Required.

- <regionName>

  The name you want for the new region. Required.

See also

virtual function, virtual signal, "Virtual Objects"

Note

Virtual regions cannot be used in the when command.

virtual save

The **virtual save** command saves the definitions of virtuals to a file.

Syntax

virtual save [-kind <kind>] [-append] [<filename>]

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

- -append

  Specifies to save **only** virtuals that are not already saved or weren't read in from a macro file. These unsaved virtuals are then appended to the specified or default file. Optional.

- <filename>

  Used for writing the virtual definitions. Optional. If you don't specify **<filename>**, the default virtual filename (*virtuals.do*) will be used. You can specify a different default in the *pref.tcl* file.

See also

virtual count, "Virtual Objects"

virtual show

The **virtual show** command lists the full path names of all explicitly defined virtuals.

Syntax

virtual show [-kind <kind>]

Arguments

- -kind <kind>

  Specifies a subset of virtuals to look at. Optional. <kind> can be implicits and explicits. Unique abbreviations are accepted.

See also

virtual define, virtual describe, "Virtual Objects"

virtual signal

The **virtual signal** command creates a new signal, known only by the GUI (not the kernel), that consists of concatenations of signals and subelements as specified in **<expressionString>**.

It cannot handle bit selects and slices of Verilog registers. Please see Concatenation of Signals or Subelements for more details on syntax.

Syntax

virtual signal [-env <path>] [-install <path>] [-delay <time>] {<expressionString>} <name>

Arguments

- -env <path>

  Specifies a hierarchical context for the signal names in **<expressionString>**, so they don't all have to be full paths. Optional.

- -install <path>

  Causes the newly-created signal to become a child of the specified region. If **-install** is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in **<expressionString>**. If the expression references more than one WLF file (dataset), the virtual signal will automatically be placed in region virtuals:/Signals. Optional.

- -delay <time>

  Specifies a value by which the virtual signal will be delayed. Optional. You can use negative values to look forward in time. If units are specified, the <time> option must be enclosed in curly braces. See the examples below for more details.

- {<expressionString>}

  A text string expression in the MTI GUI expression format that defines the signal and subelement concatenation. Can also be a literal constant or computed subexpression. Required. For details on syntax, please see *Syntax and Conventions*.

- <name>

  The name you define for the virtual signal. Required. Case is ignored unless installed in a Verilog region. Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation. If using VHDL extended identifier notation, **<name>** needs to be quoted with double quotes or with curly braces.

Examples

- Reconstruct a bus *sim:/chip/alu/a(4 downto 0)*, using VHDL notation, assuming that *a_ii* are all scalars of the same type.

```
virtual signal -env sim:/chip/alu { (concat_range (4 downto 0))(a_04
& a_03 & a_02 & a_01 & a_00) } a
```

- Reconstruct a bus *sim:chip.alu.a[4:0]*, using Verilog notation. Note that the concatenation notation starts with "&{" rather than "{".

```
virtual signal -env sim:chip.alu
{ (concat_range [4:0])&{a_04, a_03, a_02, a_01, a_00} } a
```

- Create a signal *sim:/testbench/stuff* which is a record type with three fields corresponding to the three specified signals. The example assumes */chipa/mode* is of type integer, */chipa/alu/a* is of type std_logic_vector, and */chipa/decode/inst* is a user-defined enumeration.

```
virtual signal -install sim:/testbench
{ /chipa/alu/a(19 downto 13) & /chipa/decode/inst & /chipa/mode }
stuff
```

- Create a virtual signal that is the same as */top/signalA* except it is delayed by 10 ps.

```
virtual signal -delay {10 ps} {/top/signalA} myDelayedSignalA
```

- Create a three-bit signal, *chip.address_mode*, as an alias to the specified bits.

```
virtual signal { chip.instruction[23:21] } address_mode
```

- Concatenate signals *a*, *b*, and *c* with the literal constant ’000’.

```
virtual signal {a & b & c & 3'b000} myextendedbus
```

- Add three missing bits to the bus *num*, creates a virtual signal *fullbus*, and then adds that signal to the Wave window.

```
virtual signal {num & "000"} fullbus
add wave -unsigned fullbus
```

- Reconstruct a bus that was fragmented by synthesis and is missing the lower three bits. Note that you would have to type in the actual bit names (i.e. num28, num27, etc.) represented by the … in the syntax above.

```
virtual signal { num31 & num30 & num29 & ... & num4 & num3 & "000" }
fullbus
add wave -unsigned fullbus
```

- Create a two-bit signal (with an enumerated type) based on the results of the subexpressions. For example, if *aold* equals *anew*, then the first bit is true (1). Alternatively, if *bold* does not equal *bnew*, the second bit is false (0). Each subexpression is evaluated independently.

```
virtual signal {(aold == anew) & (bold == bnew)} myequalityvector
```

- Create signal *newbus* that is a concatenation of bus1 (bit-reversed) and bus2[7:4] (bit-reversed). Assuming bus1 has indices running 7 downto 0, the result will be newbus[11:0] with the upper 8 bits being bus1[0:7] and the lower 4 bits being bus2[4:7]. See Concatenation Directives for further details.

```
virtual signal {(concat_reverse)(bus1 & bus2[7:4])} newbus
```

Commands fully compatible with virtual signals

add list	add log log	add wave
delete	describe ("virtual describe" is a little faster)	examine
find	force noforce	restart
searchlog	show	

Commands compatible with virtual signals using [virtual expand <signal>]

drivers	vcd add

Commands not currently compatible with virtual signals

when

See also

virtual count	virtual define	virtual delete
virtual describe	virtual expand	virtual hide
virtual log	virtual nohide	virtual nolog
virtual region	virtual save	virtual show
virtual function	virtual type	“Virtual Objects

virtual type

The **virtual type** command creates a new enumerated type, known only by the GUI, not the kernel. Virtual types are used to convert signal values to character strings. The command works with signed integer values up to 64 bits.

Virtual types cannot be used in the when command.

Syntax

virtual type -delete <name> | {<list_of_strings>} <name>

Arguments

- -delete <name>

  Deletes a previously defined virtual type. <name> is the name you gave the virtual type when you originally defined it. Required if not defining a type.

- {<list_of_strings>}

  A list of values and their associated character strings. Required if **-delete** is not used. Values can be expressed in decimal or based notation and can include "don't-cares" (see examples below). Three kinds of based notation are supported: Verilog, VHDL, and C-language styles. The values are interpreted without regard to the size of the bus to be mapped. Bus widths up to 64 bits are supported.

  There is currently no restriction on the contents of each string, but if strings contain spaces they would need to be quoted, and if they contain characters treated specially by Tcl (square brackets, curly braces, backslashes…), they would need to be quoted with curly braces.

  See the examples below for further syntax.

- <name>

  The user-defined name of the virtual type. Required if **-delete** is not used. Case is not ignored. Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation. If using VHDL extended identifier notation, **<name>** needs to be quoted with double quotes or with curly braces.

Examples

- Using positional notation, associates each string with an enumeration index, starting at zero and increasing by one in the positive direction.When *myConvertedSignal* is displayed in the Wave, List, or Objects window, the string "state0" will appear when *mysignal* == 0, "state1" when *mysignal* == 1, "state2" when *mysignal* == 2, etc.

  ```
  virtual type {state0 state1 state2 state3} mystateType
  virtual function {(mystateType)mysignal} myConvertedSignal
  add wave myConvertedSignal
  ```

- Use sparse mapping of bus values to alphanumeric strings for an 8-bit, one-hot encoding. It shows the variety of syntax that can be used for values. The value "default" has special meaning and corresponds to any value not explicitly specified.

```
virtual type {{0 NULL_STATE} {1 st1} {2 st2} {0x04 st3} {16'h08 st4} \
             {'h10 st5} {16#20 st6} {0b01000000 st7} {0x80 st8} \
             {default BAD_STATE}} myMappedType
virtual function {(myMappedType)mybus} myConvertedBus
add wave myConvertedBus
```

- Delete the virtual type "mystateType".

  ```
  virtual type -delete mystateType
  ```

- Create a virtual type that includes "don’t-cares" (the ’-’ character).

  ```
  virtual type {{0x01-- add}{0x02-- sub}{default bad}} mydecodetype
  ```

- Create a virtual type using a mask for "don’t-cares." The middle field is the mask, and the mask should have bits set to 1 for the bits that are don't care.

  ```
  virtual type {{0x0100 0xff add}{0x0200 0xff sub}{default bad}}
  mydecodetype
  ```

See also

virtual function, “Virtual Objects”

vlib

The **vlib** command creates a design library. You must use **vlib** rather than operating system commands to create a library directory or index file.

If the specified library already exists as a valid ModelSim library, the **vlib** command will exit with a warning message without touching the library.

Syntax

vlib [-archive [-compact <percent>]] [-format { 1 | 3 }] [-help] [-dos | -short | -unix | -long] [-lock | -unlock] [-locklib | -unlocklib <] <name>

Arguments

- -archive [-compact <percent>]

  Causes design units that are compiled into the created library to be stored in archives rather than in subdirectories. Optional. Refer to "Archives" for more details.

  You may optionally specify a decimal number between 0 and 1 that denotes the allowed percentage of wasted space before archives are compacted. By default archives are compacted when 50% (.5) of their space is wasted. See an example below.

- -format { 1 | 3 }

  Allows you to convert a library compiled with the current to be compatible with a previous release.

  1 — allows you to convert a library to be compatible with the 6.2 series and earlier.

  3 — allows you to convert a library to be compatible with the 6.3 series and newer.

- -help

  Displays the command's options and arguments. Optional.

- -dos

  Specifies that subdirectories in a library have names that are compatible with DOS. Not recommended if you use the vmake utility. Optional.

- -short

  Interchangeable with the **-dos** argument. Optional.

- -unix

  Specifies that subdirectories in a library may have long file names that are NOT compatible with DOS. Optional. Default for SE.

- -long

  Interchangeable with the **-unix** argument. Optional.

- -lock | -unlock

  Locks an existing design unit so it cannot be recompiled or refreshed. The -unlock switch reverses this action. Optional. File permissions are not affected by these switches.

- -locklib | -unlocklib <

  Locks a complete library so that compilation cannot target the library and the library cannot be refreshed. The -unlocklib switch reverses this action. Optional. File permissions are not affected by these switches.

- <name>

  Specifies the pathname or archive name of the library to be created. Required.

Examples

- Create the design library *design*. You can define a logical name for the library using the vmap command or by adding a line to the library section of the *modelsim.ini* file that is located in the same directory.

  ```
  vlib design
  ```

- Create the design library *uut* and specifies that any design units compiled into the library are created as archives. Also specifies that each archive be compacted when 30% of its space is wasted.

  ```
  vlib -archive -compact .3 uut
  ```

vlog

The **vlog** command compiles Verilog source code and SystemVerilog extensions into a specified working library (or to the **work** library by default).

The **vlog** command may be invoked from within ModelSim or from the operating system command prompt. It may also be invoked during simulation.

Compiled libraries are major-version dependent. When moving between major versions, you have to refresh compiled libraries using the **-refresh** argument to **vlog**. This is not true for minor versions (letter releases).

All arguments to the **vlog** command are case sensitive: -WORK and -work are not equivalent.

The IEEE P1800 Draft Standard for SystemVerilog requires that the default behavior of the **vlog** command is to treat each Verilog design file listed on the command line as a separate compilation unit. This behavior is a change in **vlog** from versions prior to 6.2, wherein all files in a single command line were concatenated into a single compilation unit. To treat multiple files listed within a single command line as a single compilation unit, use either the **vlog -mfcu** argument or the MultiFileCompilationUnit *modelsim.ini* file variable.

Syntax

vlog [-93] [-compat] [-compile_uselibs[=<directory_name>]] [-covercells <filename>]
[-cuname] [+define+<macro_name>[=<macro_text>]] [+delay_mode_distributed]
[+delay_mode_path] [+delay_mode_unit] [+delay_mode_zero]
[-dpiheader <filename>] [-error <msg_number>[,<msg_number>,…]] [-f <filename>]
[-force_refresh <design_unit>] [-fsmnoresettrans] [-fsmxassign]
[-genvhdl [<svb>] [f <filename>]] [-gen_xml <design_unit> <filename>] [-hazards] [-help]
[+incdir+<directory>] [-incr] [-isymfile] [+libext+<suffix>] [-libmap <pathname>]
[-libmap_verbose] [+librescan] [-line <number>] [-lint] [+maxdelays] [+mindelays]
[-mixedansiports] [-mfcu | -sfcu] [-nocovercells <filename>] [-nodbgsym] [-noincr]
[+nolibcell] [-nologo] [+nospecify] [-note <msg_number>[,<msg_number>,…]]
[+notimingchecks] [+nowarn<CODE>] [-nowarn <category_number>]
[-O0] [-quiet] [-R [<simargs>]] [-refresh] [-source] [-s] [-sv]
[-suppress <msg_number>[,<msg_number>,…]] [-time]
[-timescale <time_units>/<time_precision>] [-togglecountlimit] [-togglewidthlimit]
[+typdelays] [-u] [-v <library_file>] [-version] [-vlog01compat] [-vlog95compat]
[-warning <msg_number>[,<msg_number>,…]]
[-work <library_name>] [-y <library_directory>] <filename>

Arguments

- -93

  Specifies that the VHDL interface to Verilog modules use VHDL 1076-1993 extended identifiers to preserve case in Verilog identifiers that contain uppercase letters. Optional.

- -compat

  Disables optimizations that result in different event ordering than Verilog-XL. Optional.

  ModelSim Verilog generally duplicates Verilog-XL event ordering, but there are cases where it is inefficient to do so. Using this option does not help you find event order dependencies, but it allows you to ignore them. Keep in mind that this option does not account for all event order discrepancies, and that using this option may degrade performance. Refer to "Event Ordering in Verilog Designs" for additional information.

- -compile_uselibs[=<directory_name>]

  Locates source files specified in a **`uselib** directive (Refer to "Verilog-XL uselib Compiler Directive"), compiles those files into automatically created libraries, and updates the *modelsim.ini* file with the logical mappings to the new libraries. Optional. If a *directory_name* is not specified, ModelSim uses the name specified in the MTI_USELIB_DIR environment variable. If that variable is not set, ModelSim creates the directory *mti_uselibs* in the current working directory.

- -covercells <filename>

  Enables code coverage of modules, within the specified file, defined by 'celldefine and 'endcelldefine compiler directives, or compiled with the -v or -y arguments. Optional. Can be used to override the CoverCells compiler control variable in the modelsim.ini file.

- -cuname

  Used only in conjunction with **-mfcu**. Optional. The **-cuname** names the compilation unit being created by **vlog**. The named compilation unit can then be specified on the vsim command line, along with the <top> design unit. The purpose of doing so is to force elaboration of specified compilation unit package, thereby forcing elaboration of a necessary 'bind' statement within that compilation unit that would otherwise not be elaborated. An example of the necessary commands is:

  ```
  vlog -cuname pkg_name -mfcu file1.sv file2.sv
  vsim top pkg_name
  ```

  You need to do this only in cases where you have a 'bind' statement in a module that might otherwise not be elaborated, because no module in the design depends on that compilation unit. In other words, if a module that depends on that compilation unit exists, you don't need to force the elaboration, for it occurs automatically. Also, if you are using qverilog to compile and simulate the design, this binding issue is handled properly automatically.

- +define+<macro_name>[=<macro_text>]

  Allows you to define a macro from the command line that is equivalent to the following compiler directive:

  ```
  `define <macro_name> <macro_text>
  ```

  Optional. You can specify more than one macro with a single +**define**. For example:

  ```
  vlog +define+one=r1+two=r2+three=r3 test.v
  ```

A command line macro overrides a macro of the same name defined with the **`define** compiler directive.

- +delay_mode_distributed

  Disables path delays in favor of distributed delays. Optional. Refer to “Delay Modes” for details.

- +delay_mode_path

  Sets distributed delays to zero in favor of using path delays. Optional.

- +delay_mode_unit

  Sets path delays to zero and non-zero distributed delays to one time unit. Optional.

- +delay_mode_zero

  Sets path delays and distributed delays to zero. Optional.

- -dpiheader <filename>

  Generates a header file that may then be included in C source code for DPI import functions. Optional. Refer to “DPI Use Flow” for additional information.

- -error <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "error." Optional. Edit the error variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- -f <filename>

  Specifies a file with more command line arguments. Optional. Allows complex arguments to be reused without retyping. Allows gzipped input files. Nesting of **-f** options is allowed.

  Refer to the section "Argument Files" for more information.

- -force_refresh <design_unit>

  Forces the refresh of all specified design units. Optional. By default, the work library is updated; use **-work <library_name>**, in conjunction with **-force_refresh**, to update a different library (for example, vlog -work <your_lib_name> -force_refresh).

  When the compiler refreshes a design unit, it checks each dependency to ensure its source has not been changed and recompiled. Sometimes the tool’s dependency checking algorithm changes from release to release. This can lead to false errors during the integrity checks performed by the **-refresh** argument. An example of such a message follows:

  ```
  ** Error: (vsim-13) Recompile /u/test/dware/dware_61e_beta.dwpackages
  because /home/users/questasim/linux/../synopsys.attributes has changed.
  ```

  The **-force_refresh** argument forces the refresh of the design unit, overriding any dependency checking errors encountered by the **-refresh** argument.

  A more conservative approach to working around **-refresh** dependency checks is to recompile the source code, if it is available.

- -fsmnoresettrans

  Disables recognition of implicit asynchronous reset transitions. Optional. This has the effect of excluding asynchronous reset transitions from any coverage results.

- -fsmxassign

  Enables recognition of finite state machines (FSMs) containing X assignment. Optional. This option is used to detect FSMs if current state variable or next state variable has been assigned "X" value in a "case" statement. FSMs containing X-assign are otherwise not detectable.

- -genvhdl [<svb>] [f <filename>]

  Generates an equivalent VHDL package in the specified file. Optional.

  s = generate the equivalent stdlogic; optional

  v = generate the equivalent vl_logic; optional

  b = generate the equivalent bit; optional

  null = generate the equivalent vl_logic

  f <filename> specifies the file into which the VHDL package is generated.

- -gen_xml <design_unit> <filename>

  Produces an XML-tagged file containing the interface definition of the specified module. Optional. This option requires a two-step process where you must 1) compile <filename> into a library with **vlog** (without **-gen_xml**) then 2) execute **vlog** with the **-gen_xml** switch, for example:

  ```
  vlib work
  vlog counter.v
  vlog -gen_xml counter counter.v
  ```

- -hazards

  Detects event order hazards involving simultaneous reading and writing of the same register in concurrently executing processes. Optional. You must also specify this argument when you simulate the design with vsim. Refer to "Hazard Detection" for more details.

Note

Enabling **-hazards** implicitly enables the **-compat** argument. As a result, using this argument may affect your simulation results.

- -help

  Displays the command's options and arguments. Optional.

- +incdir+<directory>

  Specifies directories to search for files included with **`include** compiler directives. Optional. By default, the current directory is searched first and then the directories specified by the

+incdir options in the order they appear on the command line. You may specify multiple **+incdir** options as well as multiple directories separated by "+" in a single **+incdir** option.

- -incr

  Performs an incremental compile. Optional. Default. Compiles only code that has changed. For example, if you change only one module in a file containing several modules, only the changed module will be recompiled. Note however that if the compile options change, all modules are recompiled regardless if you use **-incr** or not.

- -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.

- +libext+<suffix>

  Works in conjunction with the **-y** option. Specifies file extensions for the files in a source library directory. Optional. By default the compiler searches for files without extensions. If you specify the **+libext** option, then the compiler will search for a file with the suffix appended to an unresolved name. You may specify only one **+libext** option, but it may contain multiple suffixes separated by "+". The extensions are tried in the order they appear in the **+libext** option.

- -libmap <pathname>

  Specifies a Verilog 2001 library map file. Optional. You can omit this argument by placing the library map file as the first option on the vlog invocation (e.g., *vlog top.map top.v top_cfg.v*).

- -libmap_verbose

  Displays library map pattern matching information during compilation. Optional. Use to troubleshoot problems with matching filename patterns in a library file.

- +librescan

  Scans libraries in command-line order for all unresolved modules. Optional.

- -line <number>

  Starts the compiler on the specified line in the Verilog source file. Optional. By default, the compiler starts at the beginning of the file.

- -lint

  Instructs ModelSim to perform the following lint-style checks: 1) warn when Module ports are NULL; 2) warn when assigning to an input port; 3) warn when referencing undeclared variables/nets in an instantiation; 4) warn when an index for a Verilog unpacked variable array reference is out of bounds. The warnings are reported as WARNING[8]. You can also enable this option using the Show_Lint variable in the *modelsim.ini* file.

  This argument generates additional array bounds-checking code by inserting checks for out-of-bound indexing into arrays. This functionality can slow down simulation.

- +maxdelays

  Selects maximum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

- +mindelays

  Selects minimum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

- -mixedansiports

  Permits partial port redeclarations.

- -mfcu

  Instructs the compiler to treat all files within a compilation command line as a single compilation unit. Optional. The default behavior is to treat each file listed in a command as a separate compilation unit, per the SystemVerilog standard. Prior versions concatenated the contents of the multiple files into a single compilation unit by default. You can also enable this option using the MultiFileCompilationUnit variable in the *modelsim.ini* file.

- -nocovercells <filename>

  Disables code coverage of modules, within the specified file, defined by ‘celldefine and ’endcelldefine compiler directives, or compiled with the -v or -y arguments. Optional. Can be used to override the CoverCells compiler control variable in the modelsim.ini file.

- -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 allowing you to view detailed information about design objects at the source level. Two major GUI features that use this database include source window annotation and textual dataflow.

  You should only specify this switch if you know that anyone using the library will not require this information for design analysis purposes.

- -noincr

  Disables incremental compile previously turned on with **-incr**. Optional.

- +nolibcell

  By default all modules compiled from a source library are treated as though they contain a **`celldefine** compiler directive. This option disables this default. The **`celldefine** directive only affects the PLI access routines **acc_next_cell** and **acc_next_cell_load**. Optional.

- -nologo

  Disables the startup banner. Optional.

- +nospecify

  Disables specify path delays and timing checks. Optional.

- -note <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "note." Optional. Edit the note variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- +notimingchecks

  Removes all timing check entries from the design as it is parsed. Optional.

- +nowarn<CODE>

  Disables warning messages in the category specified by <CODE>. Optional. Warnings that can be disabled include the <CODE> name in square brackets in the warning message. For example,

  ```
  ** Warning: test.v(15): [RDGN] - Redundant digits in numeric
  literal.
  ```

  This warning message can be disabled by specifying +nowarnRDGN.

- -nowarn <category_number>

  Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. Multiple **-nowarn** switches are allowed. Warnings may be disabled for all compiles via the Main window **Compile > Compile Options** menu command or the *modelsim.ini* file (refer to "VHDL Compiler Control Variables").

  The warning message categories are described in Table 2-6:

Table 2-6. Warning Message Categories for vlog -nowarn

Category number	Description
12	non-LRM compliance in order to match Cadence behavior
13	constructs that code coverage can't handle

- -O0

  Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out. Add the DisableOpt variable to [vlog] section of the *modelsim.ini* file to set a permanent default.

- -quiet

  Disables 'Loading' messages. Optional.

- -R [<simargs>]

  Instructs the compiler to invoke vsim after compiling the design. The compiler automatically determines which top-level modules are to be simulated. The command line arguments following -R are passed to the simulator, not the compiler. Place the -R option at the end of the command line or terminate the simulator command line arguments with a single "-" character to differentiate them from compiler command line arguments.

  The -R option is not a Verilog-XL option, but it is used by ModelSim to combine the compile and simulate phases together as you may be used to doing with Verilog-XL. It is not recommended that you regularly use this option because you will incur the unnecessary overhead of compiling your design for each simulation run. Mainly, it is provided to ease the transition to ModelSim.

- -refresh

  Regenerates a library image. Optional. By default, the work library is updated. To update a different library, use **-work <library_name>** with **-refresh** (for example, vlog -work <your_lib_name> -refresh). If a dependency checking error occurs which prevents the refresh, use the **vlog -force_refresh** argument. See **vlog** examples for more information. You may use a specific design name with **-refresh** to regenerate a library image for that design, but you may not use a file name.

- -sfcu

  Instructs the compiler to treat all files within a compilation command line as a separate compilation units. This is the default behavior and is the inverse of the behavior of -mfcu.

  This switch will override the MultiFileCompilationUnit variable if it is set to "1" in the *modelsim.ini* file.

- -source

  Displays the associated line of source code before each error message that is generated during compilation. Optional; by default, only the error message is displayed.

- -s

  Instructs the compiler not to load the **standard** package. Optional. This argument should only be used when you are compiling the **sv_std** package.

- -sv

  Enables SystemVerilog features and keywords. Optional. By default ModelSim follows the rules of IEEE Std 1364-2001 and ignores SystemVerilog keywords. If a source file has a ".sv" extension, ModelSim will automatically parse SystemVerilog keywords.

- -suppress <msg_number>[,<msg_number>,…]

  Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress Fatal or Internal messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing message Severity Level” for more information.

- -time

  Reports the "wall clock time" **vlog** takes to compile the design. Optional. Note that if many processes are running on the same system, wall clock time may differ greatly from the actual "cpu time" spent on **vlog**.

- -timescale <time_units>/<time_precision>

  Specifies the default timescale for modules not having an explicit timescale directive in effect during compilation. Optional. The format of the -timescale argument is the same as that of the `timescale directive. The format for <*time_units*> and <*time_precision*> is <*n*><*units*>. The value of <*n*> must be 1, 10, or 100. The value of <*units*> must be fs, ps, ns, us, ms, or s. In addition, the <*time_units*> must be greater than or equal to the <*time_precision*>.

- -togglecountlimit

  Limits the toggle coverage count for a toggle node. Optional. After the limit is reached, further activity on the node is ignored for toggle coverage. All possible transition edges must reach this count for the limit to take effect. For example, if you are collecting toggle data on 0->1 and 1->0 transitions, both transition counts must reach the limit. If you are collecting "full" data on 6 edge transitions, all 6 must reach the limit. Overrides the global value set by the ToggleCountLimit *modelsim.ini* variable.

- -togglewidthlimit

  Sets the maximum width of signals that are automatically added to toggle coverage with the **-cover t** argument. Optional. Can be set on design unit basis. Overrides the default value (128) of the ToggleWidthLimit *modelsim.ini* variable.

- +typdelays

  Selects typical delays from the "min:typ:max" expressions. Default. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

- -u

  Converts regular Verilog identifiers to uppercase. Allows case insensitivity for module names. Optional.

- -v <library_file>

  Specifies a source library file containing module and UDP definitions. Optional. Refer to "Verilog-XL Compatible Compiler Arguments" for more information.

  After all explicit filenames on the **vlog** command line have been processed, the compiler uses the **-v** option to find and compile any modules that were referenced but not yet defined. Modules and UDPs within the file are compiled only if they match previously unresolved references. Multiple **-v** options are allowed. See additional discussion in the examples.

- -version

  Returns the version of the compiler as used by the licensing tools. Optional.

- -vlog01compat

  Ensures compatibility with rules of IEEE Std 1364-2001. Default.

- -vlog95compat

  Disables Verilog 2001 keywords, which ensures that code that was valid according to the 1364-1995 spec can still be compiled. By default ModelSim follows the rules of IEEE Std 1364-2001. Some requirements in 1364-2001 conflict with requirements in 1364-1995. Optional. Edit the vlog95compat variable in the *modelsim.ini* file to set a permanent default.

- -warning <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "warning." Optional. Edit the warning variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -work <library_name>

  Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional; by default, the compiled design units are added to the **work** library. The specified pathname overrides the pathname specified for work in the project file.

- -y <library_directory>

  Specifies a source library directory containing module and UDP definitions. Optional. Refer to "Verilog-XL Compatible Compiler Arguments" for more information.

  After all explicit filenames on the **vlog** command line have been processed, the compiler uses the **-y** option to find and compile any modules that were referenced but not yet defined. Files within this directory are compiled only if the file names match the names of previously unresolved references. Multiple **-y** options are allowed. You will need to specify a file suffix by using **-y** in conjunction with the **+libext+<suffix>** option if your filenames differ from your module names. See additional discussion in the examples.

Note

Any **-y** arguments that follow a **-refresh** argument on a **vlog** command line are ignored. Any **-y** arguments that come before the **-refresh** argument on a **vlog** command line are processed.

- <filename>

  Specifies the name of the Verilog source code file to compile. One filename is required. Multiple filenames can be entered separated by spaces. Wildcards can be used.

Examples

- Compile the Verilog source code contained in the file *example.vlg*.

  ```
  vlog example.vlg
  ```

- After compiling *top.v*, **vlog** will scan the file *und1* for modules or primitives referenced but undefined in *top.v*. Only referenced definitions will be compiled.

```
vlog top.v -v und1
```

- After compiling *top.v*, **vlog** will scan the *vlog_lib* library for files with modules with the same name as primitives referenced, but undefined in *top.v*. The use of **+libext+.v+.u** implies filenames with a *.v* or *.u* suffix (any combination of suffixes may be used). Only referenced definitions will be compiled.

  ```
  vlog top.v +libext+.v+.u -y vlog_lib
  ```

  The **-work** option specifies **mylib** as the library to regenerate. **-refresh** rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of ModelSim.

- If your library contains VHDL design units, be sure to regenerate the library with the vcom command using the **-refresh** option as well. Refer to "Regenerating Your Design Libraries" for more information.

  ```
  vlog -work mylib -refresh
  ```

- The **-incr** option determines whether or not the module source or compile options have changed as *module1.v* is parsed. If no change is found, the code generation phase is skipped. Differences in compile options are determined by comparing the compiler options stored in the _info file with the compiler options given. They must match exactly

  ```
  .vlog module1.v -u -O0 -incr
  ```

- The **-timescale** option specifies the default timescale for *module1.v*, which did not have an explicit timescale directive in effect during compilation. Quotes are necessary because the argument contains white spaces.vlog module1.

  ```
  v -timescale "1 ns / 1 ps"
  ```

vmake

The **vmake** utility allows you to use a UNIX or Windows MAKE program to maintain individual libraries. You run **vmake** on a compiled design library. This utility operates on multiple source files per design unit; it supports Verilog include files as well as Verilog and VHDL PSL vunit files.

Note

If a design is spread across multiple libraries, then each library must have its own makefile and you must build each one separately.

By default, the output of **vmake** is sent to stdout—however, you can send the output to a makefile by using the shell redirect operator (>) along with the name of the file. You can then run the makefile with a version of MAKE (not supplied with ModelSim) to reconstruct the library. *This command must be invoked from either the UNIX or the Windows/DOS prompt.*

A MAKE program is included with Microsoft Visual C/C++, as well as many other program development environments.

After running the **vmake** utility, MAKE recompiles only the design units (and their dependencies) that have changed. You run **vmake** only once; then you can simply run MAKE to rebuild your design. If you add new design units or delete old ones, you should re-run **vmake** to generate a new makefile.

The **vmake** utility ignores library objects compiled with **-nodebug**.

Syntax

vmake [-du <design_unit_name>] [-f <filename>] [-fullsrcpath] [-help] [-ignore] [<library_name>]

Arguments

- -du <design_unit_name>

  Specifies that a vmake file will be generated only for the specified design unit. Optional. You can specify this argument any number of times for a single vmake command.

- -f <filename>

  Specifies a file to read command line arguments from. Optional.

- -fullsrcpath

  Produces complete source file paths within generated makefiles. Optional. By default source file paths are relative to the directory in which compiles originally occurred. This argument makes it possible to copy and evaluate generated makefiles within directories that are different from where compiles originally occurred.

- -help

  Displays the command's options and arguments. Optional.

- -ignore

  Omits a make rule for the named primary design unit and its secondary design units. Optional.

- <library_name>

  Specifies the library name; if none is specified, then **work** is assumed. Optional.

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

The **vmap** command defines a mapping between a logical library name and a directory by modifying the *modelsim.ini* file.

With no arguments, **vmap** reads the appropriate *modelsim.ini* file(s) and prints the current logical library to physical directory mappings. Returns nothing.

Syntax

vmap [-help] [-c] [-del] [<logical_name>] [<path>]

Arguments

- -help

  Displays the command's options and arguments. Optional.

- -c

  Copies the default *modelsim.ini* file from the ModelSim installation directory to the current directory. Optional.

  This argument is intended only for making a copy of the default *modelsim.ini* file to the current directory. Do not use it while making your library mappings or the mappings may end up in the incorrect copy of the *modelsim.ini*.

- -del

  Deletes the mapping specified by <logical_name> from the current project file. Optional.

- <logical_name>

  Specifies the logical name of the library to be mapped. Optional.

- <path>

  Specifies the pathname of the directory to which the library is to be mapped. Optional. If omitted, the command displays the mapping of the specified logical name.

vsim

The **vsim** command is used to invoke the VSIM simulator, to view the results of a previous simulation run (when invoked with the **-view** switch), or to view coverage data stored in the UCDB from a previous simulation run (when invoked with the **-viewcov** switch).

You can simulate a VHDL configuration or an entity/architecture pair; a Verilog module or configuration. If you specify a VHDL configuration, it is invalid to specify an architecture. During elaboration **vsim** determines if the source has been modified since the last compile.

This command may be used in batch mode from the Windows command prompt. Refer to "Batch Mode" for more information on the VSIM batch mode.

To **manually interrupt design loading** use the Break key or <Ctrl-c> from a shell.

You can invoke **vsim** from a command prompt or in the Transcript pane of the Main window. You can also invoke it from the GUI by selecting Simulate > Start Simulation.

All arguments to the **vsim** command are case sensitive: -g and -G are not equivalent.

Syntax

Note

This Syntax section presents all of the vsim switches in alphabetical order, while the Arguments section groups the arguments into the following sections:

Arguments, all languages
Arguments, VHDL
Arguments, Verilog
Arguments, object

vsim

[-absentisempty] [+alt_path_delays] [-assertfile <filename>]

[-c] [-capacity] [-colormap new]

[-debugDB=<db_pathname>] [+delayed_timing_checks] [-display <display_spec>]
[-do "<command_string>" | <macro_file_name>] [-dpiexportobj <objfile>]
[+dumpports+collapse] [+dumpports+direction]
[+dumpports+no_strength_range]
[+dumpports+unique]

[-error <msg_number>[,<msg_number>,…]]
[-errorfile <filename>]

[-f <filename>] [-fatal <msg_number>[,<msg_number>,…]]

[-g<Name>=<Value> …] [-G<Name>=<Value> …] [-gblso <filename>]
[-geometry <geometry_spec>] [-gui]

[-hazards] [-help]

[-i] [-installcolormap]

[-keeploaded] [-keeploadedrestart] [-keepstdout]

[-l <filename>] [-lib <libname>] [-L <library_name> …]
[-Lf <library_name> …] [<library_name>.<design_unit>]

[+maxdelays] [+mindelays]

[-msgmode both | tran | wlf] [-multisource_delay min | max | latest]
[+multisource_int_delays]

[-name <name>] [-noautoldlibpath] [-nodpiexports] [+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] [-noexcludehiz] [-nofileshare] [-noglitch] [+nosdferror]
[+nosdfwarn] [+nospecify] [-note <msg_number>[,<msg_number>,…]] [+notimingchecks]
[+nowarn<CODE>] [+ntc_warn] [-onfinish ask | stop | exit]

[-pli "<object list>"] [+<plusarg>] [-printsimstats]
[+pulse_e/<percent>] [+pulse_e_style_ondetect] [+pulse_e_style_onevent]
[+pulse_r/<percent>]

[-quiet]

[+sdf_iopath_to_prim_ok]
[+sdf_nocheck_celltype]
[-sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=]<sdf_filename>]
[-sdfmaxerrors <n>] [-sdfnoerror] [-sdfnowarn] [+sdf_verbose] [-std_input <filename>]
[-std_output <filename>] [+show_cancelled_e]
[-strictvital] [-suppress <msg_number>[,<msg_number>,…]] [-sv_lib <shared_obj>]
[-sv_liblist <filename>] [-sv_root <dirname>]
[-sync]

[-t [<multiplier>]<time_unit>] [-tab <tabfile>] [-tag <string>] [-title <title>] [-togglecountlimit]
[-toggleMaxIntValues <int>] [-toggleNoIntegers] [-togglenovlogints]
[-togglevlogints][-togglewidthlimit] [-trace_foreign <int>] [+transport_int_delays]
[+transport_path_delays] [+typdelays]

[-v2k_int_delays][-vcdstim [<instance>=]<filename>] [-version]
[-view [<dataset_name>=]<WLF_filename>]
[-viewcov [<dataset_name>=]<UCDB_filename>] [-visual <visual>] [-vital2.2b]

[-warning <msg_number>[,<msg_number>,…]] [-wlf <filename>] [-wlfcachesize <n>]
[-wlfcollapsedelta] [-wlfcollapsetime] [-wlfcompress] [-wlfdeleteonquit] [-wlflock]
[-wlfnocollapse] [-wlfnocompress] [-wlfnodeleteonquit] [-wlfnolock] [-wlfnoopt] [-wlfopt]
[-wlfsimcachesize <n>] [-wlfslim <size>] [-wlftlim <duration>]
[-wlfthreads | -wlfnothreads]

Arguments, all languages

- -assertfile <filename>

  Designates an alternative file for recording VHDL assertion messages. Optional. An alternate file may also be specified by the AssertFile *modelsim.ini* variable. By default, assertion messages are output to the file specified by the TranscriptFile variable in the *modelsim.ini* file (refer to "Creating a Transcript File").

- -c

  Specifies that the simulator is to be run in command-line mode. Optional. Refer to "Modes of Operation" for more information.

- -capacity

  Enables the fine-grain analysis display of memory capacity (coarse-grain analysis is enabled by default). Optional.

- -colormap new

  Specifies that the window should have a new private colormap instead of using the default colormap for the screen. Optional.

- -display <display_spec>

  Specifies the name of the display to use. Optional. Does not apply to Windows platforms.

  For example:

  ```
  -display :0
  ```

- -displaymsgmode both | tran | wlf

  Controls the transcription of $display system task messages to the transcript and/or the Message Viewer. Refer to the section "Message Viewer Tab" in the User's Manual for more information and the displaymsgmode .ini file variable.

  both — outputs messages to both the transcript and the WLF file.

  tran — outputs messages only to the transcript, therefore they are not available in the Message Viewer. Default behavior

  wlf — outputs messages only to the WLF file/Message Viewer, therefore they are not available in the transcript.

  The display system tasks displayed with this functionality include: $display, $strobe, $monitor, $write as well as the analogous file I/O tasks that write to STDOUT, such as $fwrite or $fdisplay.

- -debugDB=<db_pathname>

  Instructs ModelSim to generate database of simulation results to be used for post-sim debug. Optional. The database pathname should have a *.dbg* extension. If a database pathname is not specified, ModelSim creates a database file named *vsim.dbg* in the current directory.

- -do “<command_string>” | <macro_file_name>

  Instructs **vsim** to use the command(s) specified by <command_string> or the macro file named by <macro_file_name> rather than the startup file specified in the *.ini* file, if any. Optional. Multiple commands should be separated by semi-colons (;).

- +dumpports+collapse

  Collapses vectors (VCD id entries) in dumpports output. Optional. The default behavior can be changed by setting the DumpportsCollapse variable in the *modelsim.ini* file.

- +dumpports+direction

  Modifies the format of extended VCD files to contain direction information. Optional.

- +dumpports+no_strength_range

  Ignores strength ranges when resolving driver values for an extended VCD file. Optional. This argument is an extension to the IEEE 1364 specification. Refer to “Resolving Values” for additional information.

- +dumpports+unique

  Generates unique VCD variable names for ports in a VCD file even if those ports are connected to the same collapsed net. Optional.

- -error <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "error." Optional. Edit the error variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- -errorfile <filename>

  Designates an alternative file for recording error messages. Optional. An alternate file may also be specified by the ErrorFile modelsim.ini variable. By default, error messages are output to the file specified by the TranscriptFile variable in the *modelsim.ini* file (refer to “Creating a Transcript File”).

- -f <filename>

  Specifies a file with more **vsim** command arguments. Optional. Allows complex argument strings to be reused without retyping.

  Refer to the section "Argument Files" for more information.

- -fatal <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "fatal." Optional. Edit the fatal variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- -g<Name>=<Value> …

  Assigns a value to all specified VHDL generics and Verilog parameters that have not received explicit values in generic maps, instantiations, or via defparams (such as top-level

generics/parameters and generics/parameters that would otherwise receive their default values). Optional. Note there is no space between -g and <Name>=<Value>.

"Name" is the name of the generic/parameter, exactly as it appears in the VHDL source (case is ignored) or Verilog source. "Value" is an appropriate value for the declared data type of a VHDL generic or any legal value for a Verilog parameter. Make sure the Value you specify for a VHDL generic is appropriate for VHDL declared data types.

No spaces are allowed anywhere in the specification, except within quotes when specifying a string value. Multiple **-g** options are allowed, one for each generic/parameter.

Name may be prefixed with a relative or absolute hierarchical path to select generics in an instance-specific manner. For example, specifying -g/top/u1/tpd=20ns on the command line would affect only the *tpd* generic on the */top/u1* instance, assigning it a value of 20ns. Specifying -gu1/tpd=20ns affects the *tpd* generic on all instances named *u1*. Specifying -gtpd=20ns affects all generics named *tpd*.

If more than one -g option selects a given generic the most explicit specification takes precedence. For example,

```
vsim -g/top/ram/u1/tpd_hl=10ns -gtpd_hl=15ns top
```

This command sets *tpd_hl* to 10ns for the */top/ram/u1* instance. However, all other *tpd_hl* generics on other instances will be set to 15ns.

Limitation: In general, generics/parameters of composite type (arrays and records) cannot be set from the command line. However, you can set string arrays, std_logic vectors, and bit vectors if they can be set using a quoted string. For example,

```
-gstrgen="This is a string"
-gslv="01001110"
```

The quotation marks must make it into vsim as part of the string because the type of the value must be determinable outside of any context. Therefore, when entering this command from a shell, put single quotes (') around the string. For example:

```
-gstrgen='"This is a string"'
```

If working within the ModelSim GUI, you would enter the command as follows:

```
{-gstrgen="This is a string"}
```

- -G<Name>=<Value> …

  Same as **-g** (see above) except that it will also override generics/parameters that received explicit values in generic maps, instantiations, or via defparams. Optional. Note there is no space between -G and <Name>=<Value>. This argument is the only way for you to alter the generic/parameter, such as its length, (other than it's value) after the design has been loaded.

- -gblso <filename>

  On UNIX platforms, loads PLI/FLI shared objects with global symbol visibility. Essentially all data and functions are exported from the specified shared object and are available to be

referenced and used by other shared objects. This option may also be specified with the GlobalSharedObjectsList variable in the *modelsim.ini* file. Optional.

- -geometry <geometry_spec>

  Specifies the size and location of the main window. Optional. Where <geometry_spec> is of the form:

  ```
  WxH+X+Y
  ```

- -gui

  Starts the ModelSim GUI without loading a design and redirects the standard output (stdout) to the GUI Transcript window. Optional.

- -help

  Displays the command's options and arguments. Optional.

- -i

  Specifies that the simulator is to be run in interactive mode. Optional.

- -installcolormap

  For UNIX only. Causes **vsim** to use its own colormap so as not to hog all the colors on the display. This is similar to the -install switch on Netscape. Optional.

- -keeploaded

  Prevents the simulator from unloading/reloading any FLI/PLI/VPI shared libraries when it restarts or loads a new design. Optional. The shared libraries will remain loaded at their current positions. User application code in the shared libraries must reset its internal state during a restart in order for this to work effectively.

- -keeploadedrestart

  Prevents the simulator from unloading/reloading any FLI/PLI/VPI shared libraries during a restart. Optional. The shared libraries will remain loaded at their current positions. User application code in the shared libraries must reset its internal state during a restart in order for this to work effectively.

  We recommend using this option if you'll be doing warm restores after a restart and the user application code has set callbacks in the simulator. Otherwise, the callback function pointers might not be valid if the shared library is loaded into a new position.

- -keepstdout

  For use with foreign programs. Instructs the simulator to not redirect the stdout stream to the Main window. Optional.

- -l <filename>

  Saves the contents of the Transcript pane to <filename>. Optional. Default is taken from the TranscriptFile variable (initially set to *transcript*) in the *modelsim.ini*.

- -L <library_name> …

  Specifies the library to search for design units instantiated from Verilog and for VHDL default component binding. Refer to "Library Usage" for more information. If multiple libraries are specified, each must be preceded by the **-L** option. Libraries are searched in the order in which they appear on the command line.

- -Lf <library_name> …

  Same as **-L** but libraries are searched before 'uselib directives. Refer to "Library Usage" for more information. Optional.

- -lib <libname>

  Specifies the default working library where **vsim** will look for the design unit(s). Optional. Default is "work".

- -msgmode both | tran | wlf

  Specifies the location(s) for the simulator to output elaboration and runtime messages. Refer to the section "Message Viewer Tab" in the User's Manual for more information.

  both — outputs messages to both the transcript and the WLF file. Default behavior

  tran — outputs messages only to the transcript, therefore they are not available in the Message Viewer.

  wlf — outputs messages only to the WLF file/Message Viewer, therefore they are not available in the transcript.

- -multisource_delay min | max | latest

  Controls the handling of multiple PORT or INTERCONNECT constructs that terminate at the same port. Optional. By default, the Module Input Port Delay (MIPD) is set to the max value encountered in the SDF file. Alternatively, you may choose the min or latest of the values. If you have a Verilog design and want to model multiple interconnect paths independently, use the **+multisource_int_delays** argument.

- +multisource_int_delays

  Enables multisource interconnect delay with pulse handling and transport delay behavior. Works for both Verilog and VITAL cells. Optional.

  Use this argument when you have interconnect data in your SDF file and you want the delay on each interconnect path modeled independently. Pulse handling is configured using the **+pulse_int_e** and **+pulse_int_r** switches (described below).

  The **+multisource_int_delays** argument cannot be used if you compiled using the **-novital** argument to **vcom**. The -novital argument instructs vcom to implement VITAL functionality using VHDL code instead of accelerated code, and multisource interconnect delays cannot be implemented purely within VHDL.

- -name <name>

  Specifies the application name used by the interpreter for send commands. This does not affect the title of the window. Optional.

- -noautoldlibpath

  Disables the default internal setting of LD_LIBRARY_PATH, enabling you to set it yourself. Optional. This argument only valid for batch mode use (vsim invoked from a shell prompt, rather than inside the GUI).

- -nocapacity

  Disables the display of both coarse-grain and fine-grain analysis of memory capacity. Optional.

- -nocompress

  Causes VSIM to create uncompressed checkpoint files. Optional. This option may also be specified with the CheckpointCompressMode variable in the *modelsim.ini* file.

- +no_notifier

  Disables the toggling of the notifier register argument of all timing check system tasks. Optional. By default, the notifier is toggled when there is a timing check violation, and the notifier usually causes a UDP to propagate an X. This argument suppresses X propagation in both Verilog and VITAL for the entire design.

- +no_tchk_msg

  Disables error messages generated when timing checks are violated. Optional. For Verilog, it disables messages issued by timing check system tasks. For VITAL, it overrides the MsgOn arguments and generics.

  Notifier registers are still toggled and may result in the propagation of Xs for timing check violations.

- -note <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "note." Optional. Edit the note variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- +notimingchecks

  Disables Verilog timing checks. (It does NOT set the generic TimingChecksOn to FALSE for all VHDL Vital models with the Vital_level0 or Vital_level1 attribute. Generics with the name TimingChecksOn on non-Vital models are unaffected.) Optional. By default, Verilog timing check system tasks ($setup, $hold,…) in specify blocks are enabled. For VITAL, the timing check default is controlled by the ASIC or FPGA vendor, but most default to enabled.

- -plicompatdefault { latest | 2005 | 2001 }

  Specifies the VPI object model behavior within vsim. This switch applies globally, not to individual libraries.

  latest — This is equivalent to the "2005" argument. This is the default behavior if you do not specify this switch or if you specify the switch without an argument.

2005 — Instructs vsim to use the object models as defined in IEEE Std 1800-2005 and IEEE Std 1364-2005. You can also use "05" as an alias.

2001 — Instructs vsim to use the object models as defined in IEEE Std 1364-2001. When you specify this argument, SystemVerilog objects will not be accessible. You can also use "01" as an alias.

You can also control this behavior with the PliCompatDefault variable in the modelsim.ini file, where the -plicompatdefault argument will override the PliCompatDefault variable.

You should note that there are a few cases where the 2005 VPI object model is incompatible with the 2001 model, which is inherent in the specifications.

Refer to the appendix "Verilog PLI/VPI/DPI" in the User's Manual for more information.

- -printsimstats

  Prints the output of the **simstats** command to the screen at the end of simulation before exiting. Edit the PrintSimStats variable in the *modelsim.ini* file to set the simulation to print the simstats data by default.

- +pulse_int_e/<percent>

  Controls how pulses are propagated through interconnect delays, where <percent> is a number between 0 and 100 that specifies the error limit as a percentage of the interconnect delay. Optional. Used in conjunction with **+multisource_int_delays** (see above). This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog.

  A pulse greater than or equal to the error limit propagates to the output in transport mode (transport mode allows multiple pending transitions on an output). A pulse less than the error limit and greater than or equal to the rejection limit (see **+pulse_int_r/<percent>** below) propagates to the output as an X. If the rejection limit is not specified, then it defaults to the error limit. For example, consider an interconnect delay of 10 along with a **+pulse_int_e/80** option. The error limit is 80% of 10 and the rejection limit defaults to 80% of 10. This results in the propagation of pulses greater than or equal to 8, while all other pulses are filtered.

- +pulse_int_r/<percent>

  Controls how pulses are propagated through interconnect delays, where <percent> is a number between 0 and 100 that specifies the rejection limit as a percentage of the interconnect delay. Optional. This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog.

  A pulse less than the rejection limit is filtered. If the error limit is not specified by **+pulse_int_e** then it defaults to the rejection limit.

- -quiet

  Disable 'Loading' messages during batch-mode simulation. Optional.

- +sdf_iopath_to_prim_ok

  Prevents **vsim** from issuing an error when it cannot locate specify path delays to annotate. If you specify this argument, IOPATH statements are annotated to the primitive driving the destination port if a corresponding specify path is not found. Optional. Refer to "SDF to Verilog Construct Matching" for additional information.

- -sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=]<sdf_filename>

  Annotates VITAL or Verilog cells in the specified SDF file (a Standard Delay Format file) with minimum, typical, or maximum timing. Optional.

  The optional argument @<delayScale> scales all values by the specified value. For example, if you specify -sdfmax@1.5…, all maximum values in the SDF file will be scaled to 150% of their original value.

  The use of [<instance>=] with <sdf_filename> is also optional; it is used when the backannotation is not being done at the top level. Refer to "Specifying SDF Files for Simulation".

- -sdfmaxerrors <n>

  Controls the number of Verilog SDF missing instance messages that will be emitted before terminating vsim. Optional. <n> is the maximum number of missing instance error messages to be emitted. The default number is 5.

- -sdfnoerror

  Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue. Optional.

- -sdfnowarn

  Disables warnings from the SDF reader. Optional. Refer to "VHDL Simulation" for an additional discussion of SDF.

- +sdf_verbose

  Turns on the verbose mode during SDF annotation. The Transcript pane provides detailed warnings and summaries of the current annotation as well as information including the module name, source file name and line number. Optional.

- -suppress <msg_number>[,<msg_number>,…]

  Prevents the specified message(s) from displaying. Optional. You cannot suppress Fatal or Internal messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -sync

  Executes all X server commands synchronously, so that errors are reported immediately. Does not apply to Windows platforms.

- -t [<multiplier>]<time_unit>

  Specifies the simulator time resolution. Optional. <time_unit> must be one of the following:

```
fs, ps, ns, us, ms, sec
```

The default is 1ps; the optional <multiplier> may be 1, 10 or 100. Note that there is no space between the multiplier and the unit (i.e., 10fs, not 10 fs).

If you omit the **-t** argument, the default time resolution depends on design type: in a Verilog design with ‘timescale directives, the minimum time precision is used; in Verilog designs without *any* timescale directives, or in a VHDL or mixed design, the value specified for the Resolution variable in the *modelsim.ini* file is used.

Once you’ve begun simulation, you can determine the current simulator resolution by invoking the report command with the **simulator state** option.

- -tab <tabfile>

  Specifies the location of a Synopsys VCS table 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>

  Specifies a string tag to append to foreign trace filenames. Optional. Used with the **-trace_foreign <int>** option. Used when running multiple traces in the same directory.

- -title <title>

  Specifies the title to appear for the ModelSim Main window. Optional. If omitted the current ModelSim version is the window title. Useful when running multiple simultaneous simulations. Text strings with spaces must be in quotes (e.g., "my title").

- -togglecountlimit

  Overrides the global toggle coverage count limit for toggle nodes in an entire simulation. Optional. After the limit is reached, further activity on the node will be ignored for toggle coverage. All possible transition edges must reach this count for the limit to take effect. For example, if you are collecting toggle data on 0->1 and 1->0 transitions, both transition counts must reach the limit. If you are collecting "full" data on 6 edge transitions, all 6 must reach the limit. Overrides the global value set by the ToggleCountLimit *modelsim.ini* variable.

- -togglewidthlimit

  Sets the maximum width of signals that are automatically added to toggle coverage with the **-cover t** argument for vcom or vlog. Optional. Can be set on design unit basis. Overrides the global value set by the ToggleWidthLimit *modelsim.ini* variable.

- -trace_foreign <int>

  Creates two kinds of foreign interface traces: a log of what functions were called, with the value of the arguments, and the results returned; and a set of C-language files to replay what the foreign interface side did.

The purpose of the logfile is to aid the debugging of your 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.

- +transport_int_delays

  Selects transport mode with pulse control for single-source nets (one interconnect path). Optional. By default interconnect delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through interconnect delays.

  This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog. This option works independently from **+multisource_int_delays**.

- -vcdstim [<instance>=]<filename>

  Specifies a VCD file from which to re-simulate the design. Optional. The VCD file must have been created in a previous ModelSim simulation using the vcd dumpports command. Refer to "Using Extended VCD as Stimulus" for more information.

- -version

  Returns the version of the simulator as used by the licensing tools. Optional.

- -view [<dataset_name>=]<WLF_filename>

  Specifies a wave log format (WLF) file for **vsim** to read. Allows you to use **vsim** to view the results from an earlier simulation. The Structure, Objects, Wave, and List windows can be opened to look at the results stored in the WLF file (other ModelSim windows will not show any information when you are viewing a dataset). See additional discussion in the Examples.

- -viewcov [<dataset_name>=]<UCDB_filename>

  Invokes vsim in the coverage view mode to display UCDB data.

- -visual <visual>

  Specifies the visual to use for the window. Optional. Does not apply to Windows platforms.

  Where <visual> may be:

  <class> <depth> — One of the following:

  {directcolor | grayscale | greyscale | pseudocolor | staticcolor | staticgray | staticgrey | truecolor}

  followed by:

  <depth> — Specifies how many bits per pixel are needed for the visual.

  default — Instructs the tool to use the default visual for the screen

  <number> — Specifies a visual X identifier.

best <depth> — Instructs the tool to choose the best possible visual for the specified <depth>, where:

<depth> — Specifies how many bits per pixel are needed for the visual.

- -warning <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "warning." Optional. Edit the warning variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -wlf <filename>

  Specifies the name of the wave log format (WLF) file to create. The default is *vsim.wlf*. Optional. This option may also be specified with the WLFFilename variable in the *modelsim.ini* file.

- -wlfcachesize <n>

  Specifies the size in megabytes of the WLF reader cache. Optional. By default the cache size is set to zero. WLF reader caching caches blocks of the WLF file to reduce redundant file I/O. This should have significant benefit in slow network environments. This option may also be specified with the WLFCacheSize variable in the *modelsim.ini* file.

- -wlfcollapsedelta

  Instructs ModelSim to record values in the WLF file only at the end of each simulator delta step. Any sub-delta values are ignored. May dramatically reduce WLF file size. This option may also be specified with the WLFCollapseMode variable in the *modelsim.ini* file. Default.

- -wlfcollapsetime

  Instructs ModelSim to record values in the WLF file only at the end of each simulator time step. Any delta or sub-delta values are ignored. May dramatically reduce WLF file size. This option may also be specified with the WLFCollapseMode variable in the *modelsim.ini* file. Optional.

- -wlfcompress

  Creates compressed WLF files. Default. Use **-wlfnocompress** to turn off compression. This option may also be specified with the WLFCompress variable in the *modelsim.ini* file.

- -wlfdeleteonquit

  Deletes the current simulation WLF file (*vsim.wlf*) automatically when the simulator exits. Optional. This option may also be specified with the WLFDeleteOnQuit variable in the *modelsim.ini* file.

- -wlflock

  Locks a WLF file. Optional. An invocation of ModelSim will not overwrite a WLF file that is being written by a different invocation.

- -wlfnocollapse

  Instructs ModelSim to preserve all events and event order. May result in relatively larger WLF files. This option may also be specified with the WLFCollapseMode variable in the *modelsim.ini* file. Optional.

- -wlfnocompress

  Causes **vsim** to create uncompressed WLF files. Optional. WLF files are compressed by default in order to reduce file size. This may slow simulation speed by one to two percent. You may want to disable compression to speed up simulation or if you are experiencing problems with faulty data in the resulting WLF file. This option may also be specified with the WLFCompress variable in the *modelsim.ini* file.

- -wlfnodeleteonquit

  Preserves the current simulation WLF file (vsim.wlf) when the simulator exits. Default. This option may also be specified with the WLFDeleteOnQuit variable in the *modelsim.ini* file.

- -wlfnolock

  Disables WLF file locking. Optional. This will prevent vsim from checking whether a WLF file is locked prior to opening it as well as preventing vsim from attempting to lock a WLF once it has been opened.

- -wlfnoopt

  Disables optimization of waveform display in the Wave window. Optional. This option may also be specified with the WLFOptimize variable in the *modelsim.ini* file.

- -wlfopt

  Optimizes the display of waveforms in the Wave window. Default. Optional. This option may also be specified with the WLFOptimize variable in the *modelsim.ini* file.

- -wlfsimcachesize <n>

  Specifies the size in megabytes of the WLF reader cache for the current simulation dataset only. Optional. By default the cache size is set to zero. This makes it easier to set different sizes for the WLF reader cache used during simulation and those used during post-simulation debug. WLF reader caching caches blocks of the WLF file to reduce redundant file I/O. If neither -wlfsimcachesize nor **WLFSimCacheSize** *modelsim.ini* variable are specified, the -wlfcachesize or **WLFCacheSize** settings will be used.

- -wlfslim <size>

  Specifies a size restriction in megabytes for the event portion of the WLF file. Optional. The default is infinite size (0). The <size> must be an integer.

  Note that a WLF file contains event, header, and symbol portions. The size restriction is placed on the event portion only. When ModelSim exits, the entire header and symbol portion of the WLF file is written. Consequently, the resulting file will be larger than the size specified with **-wlfslim**.

If used in conjunction with **-wlftlim**, the more restrictive of the limits takes precedence.

This option may also be specified with the WLFSizeLimit variable in the *modelsim.ini* file. (See Limiting the WLF File Size.)

- -wlfthreads | -wlfnothreads

  Specifies whether the logging of information to the WLF file is performed using multithreading.

  This behavior is on (-wlfthreads) by default on Solaris and Linux platforms where there are more than one processor on the system. If there is only one processor available, or you are running on a Windows system, this behavior is off by default (-wlfnothreads).

  When this behavior is enabled, the logging of information is performed on the secondary processor while the simulation and other tasks are performed on the primary processor.

  You can turn this option off with the -wlfnothreads option, which you may want to do if you are performing several simulations with logging at the same time.

  You can also control this behavior with the WLFUseThreads variable in the *modelsim.ini* file.

- -wlftlim <duration>

  Specifies the duration of simulation time for WLF file recording. Optional. The default is infinite time (0). The <duration> is an integer of simulation time at the current resolution; you can optionally specify the resolution if you place curly braces around the specification. For example,

  ```
  {5000 ns}
  ```

  sets the duration at 5000 nanoseconds regardless of the current simulator resolution.

  The time range begins at the current simulation time and moves back in simulation time for the specified duration. For example,

  ```
  vsim -wlftlim 5000
  ```

  writes at most the last 5000ns of the current simulation to the WLF file (the current simulation resolution in this case is ns).

  If used in conjunction with **-wlfslim**, the more restrictive of the limits will take effect.

  This option may also be specified with the WLFTimeLimit variable in the *modelsim.ini* file.

  The **-wlfslim** and **-wlftlim** switches were designed to help users limit WLF file sizes for long or heavily logged simulations. When small values are used for these switches, the values may be overridden by the internal granularity limits of the WLF file format. (See Limiting the WLF File Size.)

Arguments, VHDL

- -absentisempty

  Causes VHDL files opened for read that target non-existent files to be treated as empty, rather than ModelSim issuing fatal error messages. Optional.

- -nocollapse

  Disables the optimization of internal port map connections. Optional.

- -nofileshare

  Turns off file descriptor sharing. Optional. By default ModelSim shares a file descriptor for all VHDL files opened for write or append that have identical names.

- -noglitch

  Disables VITAL glitch generation. Optional.

  Refer to "VHDL Simulation" for additional discussion of VITAL.

- +no_glitch_msg

  Disable VITAL glitch error messages. Optional.

- -std_input <filename>

  Specifies the file to use for the VHDL TextIO STD_INPUT file. Optional.

- -std_output <filename>

  Specifies the file to use for the VHDL TextIO STD_OUTPUT file. Optional.

- -strictvital

  Specifies to exactly match the VITAL package ordering for messages and delta cycles. Optional. Useful for eliminating delta cycle differences caused by optimizations not addressed in the VITAL LRM. Using this argument negatively impacts simulator performance.

- -toggleMaxIntValues <int>

  Specifies the maximum number of VHDL integer values to record for toggle coverage. Optional. This limit variable may be changed on a per-signal basis. The default value of <int> is 100 values.

- -toggleNoIntegers

  Turns off toggle coverage recording of VHDL integer values. Optional.

- -vital2.2b

  Selects SDF mapping for VITAL 2.2b (default is VITAL 2000). Optional.

Arguments, Verilog

- +alt_path_delays

  Configures path delays to operate in inertial mode by default. Optional. In inertial mode, a pending output transition is cancelled when a new output transition is scheduled. The result is that an output may have no more than one pending transition at a time, and that pulses narrower than the delay are filtered. The delay is selected based on the transition from the cancelled pending value of the net to the new pending value. The **+alt_path_delays** option modifies the inertial mode such that a delay is based on a transition from the current output value rather than the cancelled pending value of the net. This option has no effect in

transport mode (see **+pulse_e/<percent>** and **+pulse_r/<percent>**).

- +delayed_timing_checks

  Causes timing checks to be performed on the delayed versions of input ports (used when there are negative timing check limits). Optional.

- -dpiexportobj <objfile>

  Generates the C export wrappers and associated compiled object code for your design. The C wrapper code is written to your *<work>/_dpi/* directory, so it must have the proper permissions. The object file(s) are written to whatever location you specify with the <objfile> argument.

  For Windows platforms, this is a required switch when using DPI that generates a .obj file suitable for linking into a .dll. Refer to "DPI Use Flow" for additional information.

  For Linux and UNIX platforms, this switch generates both a .o and a so file. The .o file is suitable for linking into a larger .so file, which may contain import code. The .so file can be used directly, for example as an argument to the vsim -gblso switch or as a dependent library in the link command for an import shared object.

  Once you compile the export wrapper code into a shared object or .dll, you can manually load it into the simulation using -sv_lib, or perhaps -gblso. When you do manually load the export wrapper code, you should use the -nodpiexports switch so that the simulation does not automatically generate and load the *<work>/_dpi/exportwrapper.so* file, which would cause symbol collisions.

- -hazards

  Enables event order hazard checking in Verilog modules. Optional. You must also specify this argument when you compile your design with vlog. Refer to "Hazard Detection" for more details.

Note

Enabling **-hazards** implicitly enables the **-compat** argument. As a result, using this argument may affect your simulation results.

- +maxdelays

  Selects the maximum value in min:typ:max expressions. Optional. The default is the typical value. Has no effect if you specified the min:typ:max selection at compile time.

- +mindelays

  Selects the minimum value in min:typ:max expressions. Optional. The default is the typical value. Has no effect if you specified the min:typ:max selection at compile time.

- +no_cancelled_e_msg

  Disables negative pulse warning messages. Optional. By default **vsim** issues a warning and then filters negative pulses on specify path delays. You can drive an X for a negative pulse using **+show_cancelled_e**.

- +no_neg_tchk

  Disables negative timing check limits by setting them to zero. Optional. By default negative timing check limits are enabled. This is just the opposite of Verilog-XL, where negative timing check limits are disabled by default, and they are enabled with the +neg_tchk option.

- +no_notifier

  Disables the toggling of the notifier register argument of all timing check system tasks. Optional. By default, the notifier is toggled when there is a timing check violation, and the notifier usually causes a UDP to propagate an X. This argument suppresses X propagation on timing violations for the entire design.

- +no_path_edge

  Causes ModelSim to ignore the input edge specified in a path delay. Optional. The result of this argument is that all edges on the input are considered when selecting the output delay. Verilog-XL always ignores the input edges on path delays.

- +no_pulse_msg

  Disables the warning message for specify path pulse errors. Optional. A path pulse error occurs when a pulse propagated through a path delay falls between the pulse rejection limit and pulse error limit set with the **+pulse_r** and **+pulse_e** options. A path pulse error results in a warning message, and the pulse is propagated as an X. The **+no_pulse_msg** option disables the warning message, but the X is still propagated.

- -no_risefall_delaynets

  Disables the rise/fall delay net delay negative timing check algorithm. Optional. This argument is provided to return ModelSim to its pre-6.0 behavior where violation regions must overlap in order to find a delay net solution. In 6.0 versions and later, ModelSim uses separate rise/fall delays, so violation regions need not overlap for a delay solution to be found.

- +no_show_cancelled_e

  Filters negative pulses on specify path delays so they don’t show on the output. Default. Use **+show_cancelled_e** to drive a pulse error state.

- +no_tchk_msg

  Disables error messages issued by timing check system tasks when timing check violations occur. Optional. Notifier registers are still toggled and may result in the propagation of Xs for timing check violations.

- -nodpiexports

  Instructs the command to not generate C wrapper code for DPI export task and function routines found at elaboration time. More specifically, the command does not generate the *exportwrapper.so* shared object file in *<work>/_dpi/*. For a description on when you should use this switch, refer to the section “Integrating Export Wrappers into an Import Shared Object” in the User’s Manual.

- -noexcludehiz

  Instructs ModelSim to include truth table rows that contain Hi-Z states in the coverage count. Without this argument, these rows are automatically excluded. Optional.

- +nosdferror

  Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue. Optional.

- +nosdfwarn

  Disables warnings from the SDF annotator. Optional.

- +nospecify

  Disables specify path delays and timing checks. Optional.

- +nowarn<CODE>

  Disables warning messages in the category specified by <CODE>. Optional. Warnings that can be disabled include the <CODE> name in square brackets in the warning message. For example:

  ```
  ** Warning: (vsim-3017) test.v(2): [TFMPC] - Too few port
  connections. Expected <m>, found <n>.
  ```

  This warning message can be disabled with **+nowarnTFMPC**.

- +ntc_warn

  Enables warning messages from the negative timing constraint algorithm. Optional. By default, these warnings are disabled.

  This algorithm attempts to find a set of delays for the timing check delayed net arguments such that all negative limits can be converted to non-negative limits with respect to the delayed nets. If there is no solution for this set of limits, then the algorithm sets one of the negative limits to zero and recalculates the delays. This process is repeated until a solution is found. A warning message is issued for each negative limit set to zero.

- -onfinish ask | stop | exit

  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 for user confirmation on whether to quit the simulation.
  - **stop —** stops simulation and leave the simulation kernal running
  - **exit** — exits out of the simulation without a prompt

  By default, the simulator exits in batch mode; prompts you in GUI mode. Edit the OnFinish variable in the *modelsim.ini* file to set the default operation of $finish.

- -pli "<object list>"

  Loads a space-separated list of PLI shared objects. Optional. The list must be quoted if it contains more than one object. This is an alternative to specifying PLI objects in the Veriuser entry in the *modelsim.ini* file, refer to "Simulator Control Variables". You can use environment variables as part of the path.

- +<plusarg>

  Arguments preceded with "+" are accessible by the Verilog PLI routine **mc_scan_plusargs()**. Optional.

- +pulse_e/<percent>

  Controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the error limit as a percentage of the path delay. Optional.

  A pulse greater than or equal to the error limit propagates to the output in transport mode (transport mode allows multiple pending transitions on an output). A pulse less than the error limit and greater than or equal to the rejection limit (see **+pulse_r/<percent>**) propagates to the output as an X. If the rejection limit is not specified, then it defaults to the error limit. For example, consider a path delay of 10 along with a **+pulse_e/80** option. The error limit is 80% of 10 and the rejection limit defaults to 80% of 10. This results in the propagation of pulses greater than or equal to 8, while all other pulses are filtered. Note that you can force specify path delays to operate in transport mode by using the **+pulse_e/0** option.

- +pulse_e_style_ondetect

  Selects the "on detect" style of propagating pulse errors (see **+pulse_e**). Optional. A pulse error propagates to the output as an X, and the "on detect" style is to schedule the X immediately, as soon as it has been detected that a pulse error has occurred. "on event" style is the default for propagating pulse errors (see **+pulse_e_style_onevent**).

- +pulse_e_style_onevent

  Selects the "on event" style of propagating pulse errors (see **+pulse_e**). Default. A pulse error propagates to the output as an X, and the "on event" style is to schedule the X to occur at the same time and for the same duration that the pulse would have occurred if it had propagated through normally.

- +pulse_r/<percent>

  Controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the rejection limit as a percentage of the path delay. Optional.

  A pulse less than the rejection limit is suppressed from propagating to the output. If the error limit is not specified by **+pulse_e** then it defaults to the rejection limit.

- +sdf_nocheck_celltype

  Disables the error check a for mismatch between the CELLTYPE name in the SDF file and the module or primitive name for the CELL instance. It is an error if the names do not match. Optional.

- +show_cancelled_e

  Drives a pulse error state ('X') for the duration of a negative pulse on a specify path delay. Optional. By default ModelSim filters negative pulses.

- -sv_lib <shared_obj>

  Specifies the name of the DPI shared object with no extension. Required for use with DPI import libraries. Refer to "DPI Use Flow" for additional information.

- -sv_liblist <filename>

  Specifies the name of a bootstrap file containing names of DPI shared objects to load. Optional.

- -sv_root <dirname>

  Specifies the directory name to be used as the prefix for DPI shared object lookups. Optional.

- -togglenovlogints

  Disables toggle coverage of Verilog integer types (except SystemVerilog enumeration types). Optional. Not needed unless the ToggleVlogIntegers modelsim.ini variable is set to 1 (on).

- -togglevlogints

  Enables toggle coverage of Verilog integer types. Optional. Disables the ToggleVlogIntegers modelsim.ini variable's default setting of off(0).

- +transport_path_delays

  Selects transport mode for path delays. Optional. By default, path delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through path delays. Note that this option affects path delays only, and not primitives. Primitives always operate in inertial delay mode.

- +typdelays

  Selects the typical value in min:typ:max expressions. Default. Has no effect if you specified the min:typ:max selection at compile time.

- -v2k_int_delays

  Causes interconnect delays to be visible at the load module port per the IEEE 1364-2001 spec. Optional. By default ModelSim annotates INTERCONNECT delays in a manner compatible with Verilog-XL. If you have $sdf_annotate() calls in your design that are not getting executed, add the Verilog task $sdf_done() after your last $sdf_annotate() to remove any zero-delay MIPDs that may have been created. May be used in tandem with the **+multisource_int_delays** argument (see above).

Arguments, object

The object arguments may be a [<library_name>].<design_unit>, a *.mpf* file, a *.wlf* file, or a text file. Multiple design units may be specified for Verilog modules and mixed VHDL/Verilog configurations.

- <library_name>.<design_unit>

  Specifies a library and associated design unit; multiple library/design unit specifications can be made. Optional. If no library is specified, the **work** library is used. Environment variables can be used. <design_unit> may be one of the following:

<configuration>	Specifies the VHDL configuration to simulate.
<module> …	Specifies the name of one or more top-level Verilog modules to be simulated. Optional.
<entity> [(<architecture>)]	Specifies the name of the top-level VHDL entity to be simulated. Optional. The entity may have an architecture optionally specified; if omitted the last architecture compiled for the specified entity is simulated. An entity is not valid if a configuration is specified.[1]

1. Most UNIX shells require arguments containing () to be single-quoted to prevent special parsing by the shell. See the examples below.

- <MPF_file_name>

  Opens the specified project. Optional.

- <WLF_file_name>

  Opens the specified dataset. Optional.

- <text_file_name>

  Opens the specified text file in a Source window. Optional.

Examples

- Invoke **vsim** on the entity *cpu* and assigns values to the generic parameters *edge* and *VCC*.

  ```
  vsim -gedge=’"low high"’ -gVCC=4.75 cpu
  ```

  If working within the ModelSim GUI, you would enter the command as follows:

  ```
  vsim {-gedge="low high"} -gVCC=4.75 cpu
  ```

  Instruct ModelSim to view the results of a previous simulation run stored in the WLF file *sim2.wlf*. The simulation is displayed as a dataset named *test*. Use the **-wlf** option to specify the name of the WLF file to create if you plan to create many files for later viewing.

  ```
  vsim -view test=sim2.wlf
  ```

  For example:

```
vsim -wlf my_design.i01 my_asic structure
vsim -wlf my_design.i02 my_asic structure
```

Annotate instance */top/u1* using the minimum timing from the SDF file *myasic.sdf*.

```
vsim -sdfmin /top/u1=myasic.sdf
```

Use multiple switches to annotate multiple instances:

```
vsim -sdfmin /top/u1=sdf1 -sdfmin /top/u2=sdf2 top
```

- This example searches the libraries *mylib* for *top(only)* and *gatelib* for *cache_set*. If the design units are not found, the search continues to the work library. Specification of the architecture (*only*) is optional.

```
vsim 'mylib.top(only)' gatelib.cache_set
```

- Invoke **vsim** on *test_counter* and run the simulation until a break event, then quit when it encounters a $finish task.

```
vsim -do "set PrefMain(forceQuit) 1; run -all" work.test_counter
```

vsim<info>

The **vsim<info>** commands return information about the current vsim executable.

- vsimAuth

  Returns the authorization level (PE/SE, VHDL/Verilog/PLUS).

- vsimDate

  Returns the date the executable was built, such as "Apr 10 2000".

- vsimId

  Returns the identifying string, such as "ModelSim 6.1".

- vsimVersion

  Returns the version as used by the licensing tools, such as "1999.04".

- vsimVersionString

  Returns the full vsim version string.

This same information can be obtained using the **-version** argument of the vsim command.

vsim_break

Stop (interrupt) the current simulation before it runs to completion. To stop a simulation and then resume it, use this command in conjunction with **run -continue**.

Syntax

vsim_break

Arguments

None.

Example

- Interrupt a simulation, then restart it from the point of interruption.

```
vsim_break
run -continue
```

vsource

The **vsource** command specifies an alternative file to use for the current source file.

This command is used when the current source file has been moved. The alternative source mapping exists for the current simulation only.

Syntax

vsource [<filename>]

Arguments

- <filename>

  Specifies a relative or full pathname. Optional. If filename is omitted, the source file for the current design context is displayed.

Examples

```
vsource design.vhd
vsource /old/design.vhd
```

wave

A number of **wave** commands are available to manipulate the Wave window.

The following tables summarize the available options for manipulating cursors, for zooming, and for adjusting the wave display view in the Wave window:

Table 2-7. Wave Window Commands for Cursor

Cursor Commands	Description
wave activecursor	Sets the active cursor to the specified cursor or, if no cursor is specified, reports the active cursor
wave addcursor	Adds a new cursor at specified time and returns the number of the newly added cursor
wave cursortime	Moves or reports the time of the specified cursor or, if no cursor is specified, the time of the active cursor
wave deletecursor	Deletes the specified cursor or, if no cursor is specified, the active cursor
wave seecursor	Positions the wave display such that the specified or active cursor appears at the specified percent from the left edge of the display – 0% is the left edge, 100% is the right edge.

Table 2-8. Wave Window Commands for Zooming

Zooming Commands	Description
wave zoomin	Zoom in the wave display by the specified factor. The default factor is 2.0.
wave zoomout	Zoom out the wave display by the specified factor. The default factor is 2.0.
wave zoomfull	Zoom the wave display to show the full simulation time.
wave zoomlast	Return to last zoom range.
wave zoomrange	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.

Table 2-9. 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

Table 2-9. Wave Window Commands for Controlling Display (cont.)

Display view Commands	Description
wave seecursor	Positions the wave display such that the specified or active cursor appears at the specified percent from the left edge of the display – 0% is the left edge, 100% is the right edge.
wave seetime	Positions the wave display such that the specified time appears at the specified percent from the left edge of the display – 0% is the left edge, 100% is the right edge.

Syntax

wave activecursor [-window <win>] [<cursor-num>]

wave addcursor [-window <win>] [-time <time>] [-name <name>] [-lock <0|1>]

wave configcursor [<cursor-num>] [-window <win>] [<option> [<value>]]

wave cursortime [-window <win>] [-time <time>] [<cursor-num>]

wave deletecursor [-window <win>] [<cursor-num>]

wave interrupt [-window <win>]

wave refresh [-window <win>]

wave seecursor [-window <win>] [-at <percent>] [<cursor-num>]

wave seetime [-window <win>] [-at <percent>] <time>

wave zoomin [-window <win>] [<factor>]

wave zoomout [-window <win>] [<factor>]

wave zoomfull [-window <win>]

wave zoomlast [-window <win>]

wave zoomrange [-window <win>] [<start-time>] [<end-time>]

Arguments

- [-at <percent>]

  Positions the display such that the time or cursor is the specified <percent> from the left edge of the wave display. 0% is the left edge; 100% is the right edge. Optional. Default is 50%.

- [<cursor-num>]

  Specifies a cursor number. Optional. If not specified, the active cursor is used.

- [<factor>]

  A number that specifies how much you want to zoom into or out of the wave display. Optional. Default value is 2.0.

- [-lock <0|1>]

  Specify the lock state of the cursor. Optional. Default is ’0’, unlocked.

- [-name <name>]

  Specify the name of the cursor. Optional. Default is "Cursor <n>" where <n> is the cursor number.

- <option> [<value>]

  Specify a value for the designated option. Currently supported options are -name, -time, and -lock. Optional. If no option is specified, current value of all options are reported.

- [<start-time>]
  [<end-time>]

  start-time and end-time are times that specify a zoom range. If neither number is specified, the command returns the current zoom range. If only one time is specified, then the zoom range is set to start at 0 and end at specified time.

- [-time <time>]

  Specifies a cursor time. Optional.

- [-window <win>]

  All commands default to the active Wave window unless this argument is used to specify a different Wave window. Optional.

Examples

- Either of these commands creates a zoom range with a start time of 20 ns and an end time of 100 ns.

  ```
  wave zoomrange 20ns 100ns
  wave zoomrange 20 100
  ```

- Return the name of cursor 2:

  ```
  wave configcursor 2 -name
  ```

- Name cursor 2, "reference cursor" and return that name with:

  ```
  wave configcursor 2 -name {reference cursor}
  ```

- Return the values of all wave configcursor options for cursor 2:

  ```
  wave configcursor 2
  ```

when

The **when** command instructs ModelSim to perform actions when the specified conditions are met.

For example, you can use the **when** command to break on a signal value or at a specific simulator time. Use the nowhen command to deactivate **when** commands.

Syntax

when [[-fast] [-id <id#>] [-label <label>] {<when_condition_expression>} {<command>}]

Description

The **when** command uses a **when_condition_expression** to determine whether or not to perform the action. Conditions can include VHDL signals and Verilog nets and registers. The **when_condition_expression** uses a simple restricted language (that is not related to Tcl), which permits only four operators and operands that may be either HDL object names, signame'event, or constants. ModelSim evaluates the condition every time any object in the condition changes, hence the restrictions.

Here are some additional points to keep in mind about the when command:

- The **when** command creates the equivalent of a VHDL process or a Verilog always block. It does not work like a looping construct you might find in other languages such as C.
- Virtual signals, functions, regions, types, etc. cannot be used in the **when** command. Neither can simulator state variables other than $now.
- With no arguments, **when** will list the currently active when statements and their labels (explicit or implicit).

Syntax

when [[-fast] [-id <id#>] [-label <label>] {<when_condition_expression>} {<command>}]

Embedded Commands Allowed with the -fast Argument

You can use any Tcl command as a <command>, along with any of the following **vsim** commands:

- bp, bd
- change
- disablebp, enablebp
- echo
- examine
- force, noforce
- log, nolog

- stop
- when, nowhen

Embedded Commands Not Allowed with the -fast Argument

- Any do commands
- Any Tk commands or widgets
- References to U/I state variables or tcl variables
- Virtual signals, functions, or types

Using Global Tcl Variables with the -fast Argument

Embedded commands that use global Tcl variables for passing a state between the when command and the user interface need to declare the state using the Tcl uivar command. For example, the variable i below is visible in the GUI. From the command prompt, you can display it (by entering echo $i) or modify it (for example, by entering set i 25).

```
set i 10
when -fast {clk == '0'} {
   uivar i
   set i [expr {$i - 1}]
   if {$i <= 0} {
      force reset 1 0, 0 250
   }
}
when -fast {reset == '0'} {
   uivar i
   set i 10
}
```

Additional Restrictions on the -fast Argument

Accessing channels (such as files, pipes, sockets) that were opened outside of the embedded command will not work. For example:

```
set fp [open mylog.txt w]
when -fast {bus} {
   puts $fp "bus change: [examine bus]"
}
```

The channel that $fp refers to is not available in the simulator, only in the user interface. Even using the uivar command does not work here because the value of $fp has no meaning in the context of the -fast argument.

The following method of rewriting this example opens the channel, writes to it, then closes it within the **when** command:

```
when -fast {bus} {
   set fp [open mylog.txt a]
   puts $fp "bus change: [examine bus]"
   close $fp
}
```

The following example is a little more sophisticated method of doing the same thing:

```
when -fast {$now == 0ns} {
   set fp [open mylog.txt w]
}
when -fast {bus} {
   puts $fp "bus change: [examine bus]"
}
when -fast {$now == 1000ns} {
   close $fp
}
```

The general principle is that any embedded command done using the -fast argument is global to all other commands used with the -fast argument. Here, {$now == 0ns} is a way to define Tcl processes that the -fast commands can use. These processes have the same restrictions that when bodies have, but the advantage is again speed as a proc will tend to execute faster than code in the when body itself.

It is recommended not to use virtual signals and expressions.

Arguments

- -fast

  Causes the embedded <command> to execute within the simulation kernel, which provides faster execution and reduces impact on simulation runtime performance. Optional. Limitations on using the -fast argument are described above (in "Embedded Commands Not Allowed with the -fast Argument"). Disallowed commands still work, but they slow down the simulation.

- -label <label>

  Used to identify individual **when** commands. Optional.

- -id <id#>

  Attempts to assign this id number to the when command. Optional. If the id number you specify is already used, ModelSim will return an error.

Note

Ids for when commands are assigned from the same pool as those used for the bp command. So, even if you haven't used an id number for a when command, it's possible it is used for a breakpoint.

- {<when_condition_expression>}

  Specifies the conditions to be met for the specified <command> to be executed. Required if a command is specified. The condition is evaluated in the simulator kernel and can be an object name, in which case the curly braces can be omitted. The command will be executed when the object changes value. The condition can be an expression with these operators:

Name	Operator
equals	==, =

Name	Operator
not equal	!=, /=
greater than	>
less than	<
greater than or equal	>=
less than or equal	<=
AND	&&, AND
OR	||, OR

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

The formal BNF syntax is:

```
condition ::= Name | { expression }

expression ::= expression AND relation
               | expression OR relation
               | relation
relation ::= Name = Literal
           | Name /= Literal
           | Name ' EVENT
           | ( expression )
Literal ::= '<char>' | "<bitstring>" | <bitstring>
```

The "=" operator can occur only between a Name and a Literal. This means that you cannot compare the value of two signals, i.e., Name = Name is not possible.

Tcl variables can be used in the condition expression but you must replace the curly braces ({}) with double quotes (""). This works like a macro substitution where the Tcl variables are evaluated once and the result is then evaluated as the when condition. Condition expressions are evaluated in the **vsim** kernel, which knows nothing about Tcl variables. That's why the condition expression must be evaluated in the GUI before it is sent to the **vsim** kernel. See below for an example of using a Tcl variable.

The ">", "<", ">=", and "<=" operators are the standard ones for vector types, not the overloaded operators in the std_logic_1164 package. This may cause unexpected results when comparing objects that contain values other than 1 and 0. ModelSim does a lexical comparison (position number) for values other than 1 and 0. For example:

```
0000 < 1111 ## This evaluates to true
H000 < 1111 ## This evaluates to false
001X >= 0010 ## This also evaluates to false
```

- {<command>}

  The command(s) for this argument are evaluated by the Tcl interpreter within the ModelSim GUI. Any ModelSim or Tcl command or series of commands are valid with one

exception—the run command cannot be used with the **when** command. Required if a when expression is specified. The command sequence usually contains a stop command that sets a flag to break the simulation run after the command sequence is completed. Multiple-line commands can be used.

Note

If you want to stop the simulation using a **when** command, you must use a stop command within your when statement. DO NOT use an exit command or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated.

Examples

- The **when** command below instructs the simulator to display the value of object *c* in binary format when there is a clock event, the clock is 1, and the value of *b* is 01100111. Finally, the command tells ModelSim to stop.

```
when -label when1 {clk'event and clk='1' and b = "01100111"} {
   echo "Signal c is [exa -bin c]"
   stop
}
```

- The commands below show an example of using a Tcl variable within a **when** command. Note that the curly braces ({}) have been replaced with double quotes ("").

```
set clkb_path /tb/ps/dprb_0/udprb/ucar_reg/uint_ram/clkb;
when -label when1 "$clkb_path'event and  $clkb_path ='1'" {
     echo "Detected Clk edge at path $clkb_path"
}
```

- The **when** command below is labeled *a* and will cause ModelSim to echo the message "b changed" whenever the value of the object *b* changes.

```
when -label a b {echo "b changed"}
```

- The multi-line **when** command below does not use a label and has two conditions. When the conditions are met, an echo and a stop command will be executed.

```
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 below sets a done condition and executes an echo and a stop command when the condition is reached.

The simulation will not stop (even if a **quit -f** command is used) unless a **stop** command is executed. 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
}
```

Here's another example that stops 100ns after a signal transition:

```
when {a = 1} {
  # If the 100ns delay is already set then let it go.
  if {[when -label a_100] == ""} {
    when -label a_100 { $now = 100 } {
      # delete this breakpoint then stop
      nowhen a_100
      stop
    }
  }
}
```

Time-based breakpoints

You can build time-based breakpoints into a **when** statement with the following syntax.

For absolute time (indicated by @) use:

```
when {$now = @1750 ns} {stop}
```

You can also use:

```
when {errorFlag = '1' OR $now = 2 ms} {stop}
```

This example adds 2 ms to the simulation time at which the **when** statement is first evaluated, then stops. The white space between the value and time unit is required for the time unit to be understood by the simulator.

You can also use variables, as shown in the following example:

```
set time 1000
when "\$now = $time" {stop}
```

The quotes instruct Tcl to expand the variables before calling the command. So, the **when** command sees:

```
when "$now = 1000" stop
```

Note that "$now" has the ’$’ escaped. This prevents Tcl from expanding the variable, because if it did, you would get:

```
when "0 = 1000" stop
```

See also

bp, disablebp, enablebp, nowhen

where

The **where** command displays information about the system environment. This command is useful for debugging problems where ModelSim cannot find the required libraries or support files. mg67000

The command displays two results on consecutive lines:

- current directory

  This is the current directory that ModelSim was invoked from, or that was specified on the ModelSim command line.

- current project file

  This is the *.mpf* file ModelSim is using. All library mappings are taken from here when a project is open. If the design is not loaded through a project, this line displays the modelsim.ini file in the current directory.

Syntax

where

Arguments

- None.

Examples

- Design is loaded through a project:

```
VSIM> where
#  Current directory is: D:\Client
#  Project is: D:/Client/monproj.mpf
```

- Design is loaded with no project (indicates the modelsim.ini file is under the `mydesign` directory):

```
VSIM> where
#  Current directory is: C:\Client\testcase\mydesign
#  Project is: modelsim.ini
```

wlf2log

The **wlf2log** command translates a ModelSim WLF file (*vsim.wlf*) to a QuickSim II logfile.

The command reads the *vsim.wlf* WLF file generated by the **add list**, **add wave**, or **log** commands in the simulator and converts it to the QuickSim II logfile format.

Note

This command should be invoked only after you have stopped the simulation using **quit -sim** or **dataset close sim**.

Syntax

wlf2log [-bits] [-fullname] [-help] [-inout] [-input] [-internal] [-l <instance_path>] [-lower] [-o <outfile>] [-output] [-quiet] <wlffile>

Arguments

- -bits

  Forces vector nets to be split into 1-bit wide nets in the log file. Optional.

- -fullname

  Shows the full hierarchical pathname when displaying signal names. Optional.

- -help

  Displays a list of command options with a brief description for each. Optional.

- -inout

  Lists only the inout ports. Optional. This may be combined with the -input, -output, or -internal switches.

- -input

  Lists only the input ports. Optional. This may be combined with the -output, -inout, or -internal switches.

- -internal

  Lists only the internal signals. Optional. This may be combined with the -input, -output, or -inout switches.

- -l <instance_path>

  Lists the signals at or below the specified HDL instance path within the design hierarchy. Optional.

- -lower

  Shows all logged signals in the hierarchy. Optional. When invoked without the -lower switch, only the top-level signals are displayed.

- -o <outfile>

  Directs the output to be written to the file specified by <outfile>. Optional. The default destination for the logfile is standard out.

- -output

  Lists only the output ports. Optional. This may be combined with the -input, -inout, or -internal switches.

- -quiet

  Disables error message reporting. Optional.

- <wlffile>

  Specifies the ModelSim WLF file that you are converting. Required.

wlf2vcd

The **wlf2vcd** command translates a ModelSim WLF file to a standard VCD file. Complex data types that are unsupported in the VCD standard (records, memories, etc.) are not converted.

Note

This command should be invoked only after you have stopped the simulation using **quit -sim** or **dataset close sim**.

Syntax

wlf2vcd [-help] [-o <outfile>] [-quiet] <wlffile>

Arguments

- -help

  Displays a list of command options with a brief description for each. Optional.

- -o <outfile>

  Specifies a filename for the output. By default, the VCD output goes to stdout. Optional.

- -quiet

  Disables warning messages that are produced when an unsupported type (e.g., records) is encountered in the WLF file. Optional.

- <wlffile>

  Specifies the ModelSim WLF file that you are converting. Required.

wlfman

The **wlfman** command allows you to get information about and manipulate WLF files.

The command performs four functions depending on which mode you use:

- **wlfman info** generates file information, resolution, versions, etc.
- **wlfman items** generates a list of HDL objects (i.e., signals) from the source WLF file and outputs it to stdout. When redirected to a file, the output is called an object_list_file, and it can be read in by **wlfman filter**. The object_list_file is a list of objects, one per line. Comments start with a '#' and continue to the end of the line. Wildcards are legal in the leaf portion of the name. Here is an example:

  ```
  /top/foo     # signal foo
  /top/u1/*    # all signals under u1
  /top/u1      # same as line above
  -r /top/u2   # recursively, all signals under u2
  ```

  Note that you can produce these files from scratch but be careful with syntax. **wlfman items** always creates a legal object_list_file.
- **wlfman filter** reads in a WLF file and optionally an object_list_file and writes out another WLF file containing filtered information from those sources. You determine the filtered information with the arguments you specify.
- **wlfman profile** generates a report of the estimated percentage of file space that each signal is taking in the specified WLF file. This command can identify signals that account for a large percentage of the WLF file size (e.g., a logged memory that uses a zero-delay integer loop to initialize the memory). You may be able to drastically reduce WLF file size by not logging those signals.
- **wlfman merge** combines two WLF files with different signals into one WLF file. It *does not* combine wlf files containing the same signals at different runtime ranges (i.e., mixedhdl_0ns_100ns.wlf & mixedhdl_100ns_200ns.wlf).

The different modes are intended to be used together. For example, you might run **wlfman profile** and identify a signal that accounts for 50% of the WLF file size. If you don't actually need that signal, you can then run **wlfman filter** to remove it from the WLF file.

Syntax

wlfman info <wlffile>

wlfman items [-n] [-v] <wlffile>

wlfman filter [-begin <time>] [-end <time>] [-f <object_list_file>] [-r <object>]
[-s <symbol>] [-t <resolution>] -o <outwlffile> <sourcewlffile>

wlfman profile [-rank] [-top <number>] <wlffile>

wlfman merge [-noopt] [-opt] -o <outwlffile> [<wlffile1> <wlffile2>]

Arguments for wlfman info

- <wlffile>

  Specifies the WLF file from which you want information. Required.

Arguments for wlfman items

- -n

  Lists regions only (no signals). Optional.

- -v

  Produces verbose output that lists the object type next to each object. Optional.

- <wlffile>

  Specifies the WLF file for which you want a profile report. Required.

Arguments for wlfman filter

- -begin <time>

  Specifies the simulation time at which you want to begin reading information from the source WLF file. Optional. By default the output includes the entire time that is recorded in the source WLF file.

- -end <time>

  Specifies the simulation time at which you want to end reading information from the source WLF file. Optional.

- -f <object_list_file>

  Specifies an object_list_file created by **wlfman items** to include in the output WLF file. Optional.

- -r <object>

  Specifies an object (region) to recursively include in the output. If <object> is a signal, the output would be the same as using **-s**. Optional.

- -s <symbol>

  Specifies an object to include in the output. Optional. By default all objects are output.

- -t <resolution>

  Specifies the time resolution of the new WLF file. Optional. By default the resolution is the same as the source WLF file.

- -o <outwlffile>

  Specifies the name of the output WLF file. Required. The output WLF file will contain all objects specified by **-f <object_list_file>**, **-r <object>**, and **-s <symbol>**. Output WLF files are always written in the latest WLF version regardless of the source WLF file version.

- <sourcewlffile>

  Specifies the source WLF file from which you want objects. Required.

Arguments for wlfman profile

- -rank

  Sorts the report by percentage. Optional.

- -top <number>

  Filters the report so that only the top <number> signals in terms of file space percentage are displayed. Optional.

- <wlffile>

  Specifies the WLF file from which you want object information. Required.

Arguments for wlfman merge

- -noopt

  Disables WLF file optimizations when writing output WLF file. Optional.

- -opt

  Forces WLF file optimizations when writing output WLF file. Optional. Default.

- -o <outwlffile>

  Specifies the name of the output WLF file. Required. The output WLF file will contain all objects from <wlffile1> and <wlffile2>. Output WLF files are always written in the latest WLF version regardless of the source WLF file version.

- <wlffile1> <wlffile2>

  Specifies the WLF files whose objects you want to copy into one WLF file. Optional.

Examples

- The output from this command would look something like this:

```
wlfman profile -rank top_vh.wlf
```

```
#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                            /top_vh/processor/data
    1                            /top_vh/cache/pdata
    1                            /top_vh/cache/gen__0/s/data
    1                            /top_vh/cache/gen__1/s/data
    1                            /top_vh/cache/gen__2/s/data
    1                            /top_vh/cache/gen__3/s/data
    2          1224      18 %    /top_vh/ptrans
    3          1216      18 %    /top_vh/sdata
    3                            /top_vh/cache/sdata
    3                            /top_vh/memory/data
    4           675      10 %    /top_vh/strans
    5           423       6 %    /top_vh/cache/gen__3/s/data_out
    6           135       3 %    /top_vh/paddr.
.
.
.
```

- wlfman profile -top 3 top_vh.wlf

The output from this command would look something like this:

```
# ID     Transitions   File %   Name
#-----   -----------   ------   ------------------------------------
    1          2192      33 %    /top_vh/pdata
    1                            /top_vh/processor/data
    1                            /top_vh/cache/pdata
    1                            /top_vh/cache/gen__0/s/data
    1                            /top_vh/cache/gen__1/s/data
    1                            /top_vh/cache/gen__2/s/data
    1                            /top_vh/cache/gen__3/s/data
    2          1224      18 %    /top_vh/ptrans
    3          1216      18 %    /top_vh/sdata
    3                            /top_vh/cache/sdata
    3                            /top_vh/memory/data
```

See also

"Recording Simulation Results With Datasets"

wlfrecover

The **wlfrecover** tool attempts to "repair" WLF files that are incomplete due to a crash or the file being copied prior to completion of the simulation. You can run the tool from the VSIM> or ModelSim> prompt or from a shell.

Syntax

wlfrecover <filename> [-force] [-q]

Arguments

- <filename>

  Specifies the WLF file to repair. Required.

- -force

  Disregards file locking and attempts to repair the file. Optional.

- -q

  Hides all messages unless there is an error while repairing the file. Optional.

write format

The **write format** command records the names and display options of the HDL objects currently being displayed in the List or Wave window.

The file created is primarily a list of add listor add wave commands, though a few other commands are included (see "Output" below). This file may be invoked with the do command to recreate the List or Wave window format on a subsequent simulation run.

When you load a wave or list format file, ModelSim verifies the existence of the datasets required by the format file. ModelSim displays an error message if the requisite datasets do not all exist. To force the execution of the wave or list format file even if all datasets are not present, use the **-force** switch with your **do** command. For example:

```
VSIM> do wave.do -force
```

Note that this will result in error messages for signals referencing nonexistent datasets. Also, -force is recognized by the format file not the **do** command.

Syntax

write format list | wave <filename>

Arguments

- list | wave

  Specifies that the contents of either the List or the Wave window are to be recorded. Required.

- <filename>

  Specifies the name of the output file where the data is to be written. Required.

Examples

- Save the current data in the List window in a file named *alu_list.do*.

  ```
  write format list alu_list.do
  ```

- Save the current data in the Wave window in a file named *alu_wave.do*.

  ```
  write format wave alu_wave.do
  ```

Output

- Below is an example of a saved Wave window format file.

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -format Logic /cntr_struct/ld
add wave -noupdate -format Logic /cntr_struct/rst
add wave -noupdate -format Logic /cntr_struct/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, five signals are added with the *-noupdate* argument to the default window pane. The **TreeUpdate** command then refreshes all five waveforms. The second **WaveActivateNextPane** command creates a second pane which contains three signals.The **WaveRestoreCursors** command restores any cursors you set during the original simulation, and the **WaveRestoreZoom** command restores the Zoom range you set. These four commands are used only in saved Wave format files; therefore, they are not documented elsewhere.

See also

add list, add wave

write list

The **write list** command records the contents of the List window in a list output file.

This file contains simulation data for all HDL objects displayed in the List window: VHDL signals and variables and Verilog nets and registers.

Syntax

write list [-events] <filename>

Arguments

- -events

  Specifies to write print-on-change format. Optional. Default is tabular format.

- <filename>

  Specifies the name of the output file where the data is to be written. Required.

Examples

- Save the current data in the List window in a file named *alu.lst*.

```
write list alu.lst
```

See also

write tssi

write preferences

The **write preferences** command saves the current GUI preference settings to a Tcl preference file. Settings saved include Wave, Objects, and Locals window column widths; Wave, Objects, and Locals window value justification; and Wave window signal name width.

Syntax

write preferences <preference file name>

Arguments

- <preference file name>

  Specifies the name for the preference file. Optional. If the file is named *modelsim.tcl*, ModelSim will read the file each time **vsim** is invoked. To use a preference file other than *modelsim.tcl* you must specify the alternative file name with the MODELSIM_TCL environment variable.

See also

You can modify variables by editing the preference file with the ModelSim notepad:

```
notepad <preference file name>
```

write report

The **write report** command prints a summary of the design being simulated including a list of all design units (VHDL configurations, entities, and packages, and Verilog modules) with the names of their source files. The summary includes a list of all source files used to compile the given design.

Syntax

write report [-capacity [-l | -s] [-assertions | -classes | -cvg | -qdas | -solver]] | [-l | -s] [-tcl] [<filename>]

Arguments

- <filename>

  Specifies the name of the output file where the data is to be written. Optional. If the <filename> is omitted, the report is written to the Transcript pane.

- -capacity

  Reports data on memory usage of various types of SystemVerilog constructs in the design. Optional. ModelSim collects memory usage data for assertions, classes, covergroups, dynamic objects, and the solver. Each of these design object types has a switch that you can specify along with -capacity in order to display its memory data. To display memory data for all object types, specify -capacity -l.

- -assertions

  Reports memory usage data for SystemVerilog assertions and cover directives.

- -classes

  Reports memory usage data for the current number of objects allocated, the current memory allocated for class object, the peak memory allocated and peak time.

- -cvg

  Reports memory usage data for the number of covergroups, cross, bins and memory allocated.

- -qdas

  Reports memory usage data for queues, dynamic arrays, and associative arrays.

- -solver

  Reports memory usage data for calls to randomize() and memory usage.

- -l

  Generates more detailed information about the design, including a list of sparse memories or the memory capacity for all object types. Default.

- -s

  Generates a short list of design information. Optional.

- -tcl

  Generates a Tcl list of design unit information. Optional. This argument cannot be used with a filename.

Examples

- Save information about the current design in a file named *alu_rpt.txt*.

  ```
  write report alu_rpt.txt
  ```

- Display a short list of information regarding the memory capacity for covergroups in the design during the simulation so far.

  ```
  write report -capacity -s cvg
  ```

- Display information on all of the calls to randomize() made during simulation so far, along with the memory usage of those calls, number of calls, and callsite information.

  ```
  write report -capacity -solver
  ```

write timing

The **write timing** command 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]

Arguments

- -recursive

  Generates timing information for the specified instance and all instances underneath it in the design hierarchy. Optional.

- -file <filename>

  Specifies the name of the output file where the data is to be written. Optional. If the -file argument is omitted, timing information is written to the Transcript pane.

- <instance_name1>…<instance_nameN>

  The name(s) of the instance(s) for which timing information will be written. Required.

- -simvalues

  Displays optimization-adjusted values for delay net delays. Optional.

Examples

- Write timing about */top/u1* and all instances underneath it in the hierarchy to the file *timing.txt*.

  ```
  write timing -r -f timing.txt /top/u1
  ```

- Write timing information about the designated instances to the Transcript pane.

  ```
  write timing /top/u1 /top/u2 /top/u3 /top/u8
  ```

write transcript

The **write transcript** command writes the contents of the Transcript pane to the specified file. The resulting file can be used to replay the transcribed commands as a DO file (macro).

The command cannot be used in batch mode. In batch mode use the standard Transcript file or redirect stdout.

Syntax

write transcript [<filename>]

Arguments

- <filename>

  Specifies the name of the output file where the data is to be written. Optional. If the <filename> is omitted, the transcript is written to a file named *transcript*.

See also

do

write tssi

The **write tssi** command records the contents of the List window in a "TSSI format" file.

The file contains simulation data for all HDL objects displayed in the List window that can be converted to TSSI format (VHDL signals and Verilog nets). A signal definition file is also generated.

The List window needs to be using symbolic radix in order for **write tssi** to produce useful output.

Syntax

write tssi <filename>

Arguments

- <filename>

  Specifies the name of the output file where the data is to be written. Required.

Description

If the <filename> has a file extension (e.g., *listfile.lst*), then the definition file is given the same file name with the extension .def (e.g., *listfile.def*). The values in the listfile are produced in the same order that they appear in the List window. The directionality is determined from the port type if the object is a port, otherwise it is assumed to be bidirectional (mode INOUT).

Objects that can be converted to SEF are VHDL enumerations with 255 or fewer elements and Verilog nets. The enumeration values U, X, 0, 1, Z, W, L, H and - (the enumeration values defined in the IEEE Standard 1164 **std_ulogic** enumeration) are converted to SEF values according to the table below. Other values are converted to a question mark (?) and cause an error message. Though the **write tssi** command was developed for use with **std_ulogic**, any signal which uses only the values defined for **std_ulogic** (including the VHDL standard type **bit**) will be converted.

std_ulogic State Characters	**SEF State Characters**		
	Input	**Output**	**Bidirectional**
U	N	X	?
X	N	X	?
0	D	L	0
1	U	H	1
Z	Z	T	F
W	N	X	?
L	D	L	0
H	U	H	1

std_ulogic State Characters	SEF State Characters		
	Input	**Output**	**Bidirectional**
-	N	X	?

Bidirectional logic values are not converted because only the resolved value is available. The TSSI TDS ASCII In Converter and ASCII Out Converter can be used to resolve the directionality of the signal and to determine the proper forcing or expected value on the port. Lowercase values x, z, w, l, and h are converted to the same values as the corresponding capitalized values. Any other values will cause an error message to be generated the first time an invalid value is detected on a signal, and the value will be converted to a question mark (?).

Note

The TDS ASCII In Converter and ASCII Out Converter are part of the TDS software. ModelSim outputs a vector file, and TSSI tools determine whether the bidirectional signals are driving or not.

See also

tssi2mti

write wave

The **write wave** command records the contents of the Wave window in PostScript format.

The output file can then be printed on a PostScript printer.

Syntax

write wave[-width <real_num>] [-height <real_num>]
[-margin <real_num>] [-start <time>] [-end <time>] [-perpage <time>] [-landscape]
[-portrait] <filename>

Arguments

- -width <real_num>

  Specifies the paper width in inches. Optional. Default is 8.5.

- -height <real_num>

  Specifies the paper height in inches. Optional. Default is 11.0.

- -margin <real_num>

  Specifies the margin in inches. Optional. Default is 0.5.

- -start <time>

  Specifies the start time (on the waveform timescale) to be written. Optional.

- -end <time>

  Specifies the end time (on the waveform timescale) to be written. Optional.

- -perpage <time>

  Specifies the time width per page of output. Optional.

- -landscape

  Use landscape (horizontal) orientation. Optional. This is the default orientation.

- -portrait

  Use portrait (vertical) orientation. Optional. The default is landscape (horizontal).

- <filename>

  Specifies the name of the PostScript output file. Required.

Examples

- Save the current data in the Wave window in a file named *alu.ps*.

  ```
  write wave alu.ps
  ```

- Write two separate pages to *top.ps*. The first page contains data from 600ns to 700ns, and the second page contains data from 701ns to 800ns.

  ```
  write wave -start 600ns -end 800ns -perpage 100ns top.ps
  ```

To make the job of creating a PostScript waveform output file easier, use the **File** > **Print Postscript** menu selection in the Wave window.

Index

— D —

— E —

— F —

— G —

— H —

— N —

— O —

— P —

— Q —

— R —

— S —

— T —

— W —

— X —

— Y —

— Z —

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/terms_conditions/enduser.cfm

IMPORTANT INFORMATION

USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE SOFTWARE. USE OF SOFTWARE INDICATES YOUR COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software between you, the end user, as an authorized representative of the company acquiring the license, and Mentor Graphics Corporation and Mentor Graphics (Ireland) Limited acting directly or through their subsidiaries (collectively "Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by you and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If you do not agree to these terms and conditions, promptly return or, if received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **GRANT OF LICENSE.** The software programs, including any updates, modifications, revisions, copies, documentation and design data ("Software"), are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to you, subject to payment of appropriate license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form; (b) for your internal business purposes; (c) for the license term; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions.

2. **EMBEDDED SOFTWARE.** If you purchased a license to use embedded software development ("ESD") Software, if applicable, Mentor Graphics grants to you a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESD compilers, including the ESD run-time libraries distributed with ESD C and C++ compiler Software that are linked into a composite program as an integral part of your compiled computer program, provided that you distribute these files only in conjunction with your compiled computer program. Mentor Graphics does NOT grant you any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into your products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

3. **BETA CODE.** Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to you a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and your use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form. If Mentor Graphics authorizes you to use the Beta Code, you agree to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. You will contact Mentor Graphics periodically during your use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of your evaluation and testing, you will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements. You agree that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on your feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this section 3 shall survive the termination or expiration of this Agreement.

4. **RESTRICTIONS ON USE.** You may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. You shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. You shall not make Software available in any form to any person other than employees and on-site contractors, excluding Mentor Graphics' competitors, whose job performance requires access and who are under obligations of confidentiality. You shall take appropriate action to protect the confidentiality of Software and ensure that any person permitted access to Software does not disclose it or use it except as permitted by this Agreement. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, you shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive from Software any source code. You may not sublicense, assign or otherwise transfer Software, this Agreement or the rights under it, whether by operation of law or otherwise ("attempted transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable transfer charges. Any attempted transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and licenses granted under this Agreement. The terms of this Agreement, including without limitation, the licensing and assignment provisions shall be binding upon your successors in interest and assigns. The provisions of this section 4 shall survive the termination or expiration of this Agreement.

5. **LIMITED WARRANTY.**

   5.1. Mentor Graphics warrants that during the warranty period Software, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Software will meet your requirements or that operation of Software will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. You must notify Mentor Graphics in writing of any nonconformity within the warranty period. This warranty shall not be valid if Software has been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND YOUR EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF SOFTWARE TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF SOFTWARE THAT DOES NOT MEET THIS LIMITED WARRANTY, PROVIDED YOU HAVE OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) SOFTWARE WHICH IS LICENSED TO YOU FOR A LIMITED TERM OR LICENSED AT NO COST; OR (C) EXPERIMENTAL BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

   5.2. THE WARRANTIES SET FORTH IN THIS SECTION 5 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO SOFTWARE OR OTHER MATERIAL PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

6. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT PAID BY YOU FOR THE SOFTWARE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 6 SHALL SURVIVE THE EXPIRATION OR TERMINATION OF THIS AGREEMENT.

7. **LIFE ENDANGERING ACTIVITIES.** NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF SOFTWARE IN ANY APPLICATION WHERE THE FAILURE OR INACCURACY OF THE SOFTWARE MIGHT RESULT IN DEATH OR PERSONAL INJURY. THE PROVISIONS OF THIS SECTION 7 SHALL SURVIVE THE EXPIRATION OR TERMINATION OF THIS AGREEMENT.

8. **INDEMNIFICATION.** YOU AGREE TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE, OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH YOUR USE OF SOFTWARE AS

DESCRIBED IN SECTION 7. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE EXPIRATION OR TERMINATION OF THIS AGREEMENT.

9. **INFRINGEMENT.**

9.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against you alleging that Software infringes a patent or copyright or misappropriates a trade secret in the United States, Canada, Japan, or member state of the European Patent Office. Mentor Graphics will pay any costs and damages finally awarded against you that are attributable to the infringement action. You understand and agree that as conditions to Mentor Graphics' obligations under this section you must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to defend or settle the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

9.2. If an infringement claim is made, Mentor Graphics may, at its option and expense: (a) replace or modify Software so that it becomes noninfringing; (b) procure for you the right to continue using Software; or (c) require the return of Software and refund to you any license fee paid, less a reasonable allowance for use.

9.3. Mentor Graphics has no liability to you if infringement is based upon: (a) the combination of Software with any product not furnished by Mentor Graphics; (b) the modification of Software other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of Software as part of an infringing process; (e) a product that you make, use or sell; (f) any Beta Code contained in Software; (g) any Software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by you that is deemed willful. In the case of (h) you shall reimburse Mentor Graphics for its attorney fees and other costs related to the action upon a final judgment.

9.4. THIS SECTION IS SUBJECT TO SECTION 6 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS AND YOUR SOLE AND EXCLUSIVE REMEDY WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY SOFTWARE LICENSED UNDER THIS AGREEMENT.

10. **TERM.** This Agreement remains effective until expiration or termination. This Agreement will immediately terminate upon notice if you exceed the scope of license granted or otherwise fail to comply with the provisions of Sections 1, 2, or 4. For any other material breach under this Agreement, Mentor Graphics may terminate this Agreement upon 30 days written notice if you are in material breach and fail to cure such breach within the 30 day notice period. If Software was provided for limited term use, this Agreement will automatically expire at the end of the authorized term. Upon any termination or expiration, you agree to cease all use of Software and return it to Mentor Graphics or certify deletion and destruction of Software, including all copies, to Mentor Graphics' reasonable satisfaction.

11. **EXPORT.** Software is subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products, information about the products, and direct products of the products to certain countries and certain persons. You agree that you will not export any Software or direct product of Software in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

12. **RESTRICTED RIGHTS NOTICE.** Software was developed entirely at private expense and is commercial computer software provided with RESTRICTED RIGHTS. Use, duplication or disclosure by the U.S. Government or a U.S. Government subcontractor is subject to the restrictions set forth in the license agreement under which Software was obtained pursuant to DFARS 227.7202-3(a) or as set forth in subparagraphs (c)(1) and (2) of the Commercial Computer Software - Restricted Rights clause at FAR 52.227-19, as applicable. Contractor/manufacturer is Mentor Graphics Corporation, 8005 SW Boeckman Road, Wilsonville, Oregon 97070-7777 USA.

13. **THIRD PARTY BENEFICIARY.** For any Software under this Agreement licensed by Mentor Graphics from Microsoft or other licensors, Microsoft or the applicable licensor is a third party beneficiary of this Agreement with the right to enforce the obligations set forth herein.

14. **AUDIT RIGHTS.** You will monitor access to, location and use of Software. With reasonable prior notice and during your normal business hours, Mentor Graphics shall have the right to review your software monitoring system and reasonably relevant records to confirm your compliance with the terms of this Agreement, an addendum to this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet report log files that you shall capture and provide at Mentor Graphics' request. Mentor Graphics shall treat as confidential information all of your information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement or addendum to this Agreement. The provisions of this section 14 shall survive the expiration or termination of this Agreement.

15. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** THIS AGREEMENT SHALL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF OREGON, USA, IF YOU ARE LOCATED IN NORTH OR SOUTH AMERICA, AND THE LAWS OF IRELAND IF YOU ARE LOCATED OUTSIDE OF NORTH OR SOUTH AMERICA. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia (except for Japan) arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the Chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section 15. This section shall not restrict Mentor Graphics' right to bring an action against you in the jurisdiction where your place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

16. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

17. **PAYMENT TERMS AND MISCELLANEOUS.** You will pay amounts invoiced, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Some Software may contain code distributed under a third party license agreement that may provide additional rights to you. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 060210, Part No. 227900