ModelSim®

SE

Command Reference

Version 5.8c

Published: 5/Mar/04

The world's most popular HDL simulator

Model Technology
8005 Boeckman Road, Bldg. E4
Wilsonville, OR 97070 USA

phone: (503) 685-0820
fax: (503) 685-0910
e-mail: support@model.com, sales@model.com
home page: http://www.model.com
support page: http://www.model.com/support

Technical support and updates

Support

Model Technology online and email technical support options, maintenance renewal, and links to international support contacts:
www.model.com/support/default.asp

Mentor Graphics support:
www.mentor.com/supportnet

Updates

Access to the most current version of ModelSim:
www.model.com/downloads/default.asp

Latest version email

Place your name on our list for email notification of news and updates:
www.model.com/products/informant.asp

Where to find our documentation

ModelSim documentation is available from our website at www.model.com/support or in the following formats and locations:

Document	Format	How to get it
ModelSim SE Installation & Licensing Guide	paper	shipped with Model*Sim*
	PDF	select **Main window > Help > SE Documentation**; also available from the Support page of our web site: www.model.com
ModelSim SE Quick Guide (command and feature quick-reference)	paper	shipped with Model*Sim*
	PDF	select **Main window > Help > SE Documentation**, also available from the Support page of our web site: www.model.com
ModelSim SE Tutorial	PDF, HTML	select **Main window > Help > SE Documentation**; also available from the Support page of our web site: www.model.com
ModelSim SE User's Manual	PDF, HTML	select **Main window > Help > SE Documentation**
ModelSim SE Command Reference	PDF, HTML	select **Main window > Help > SE Documentation**
Foreign Language Interface Reference	PDF, HTML	select **Main window > Help > SE Documentation**
Std_DevelopersKit User's Manual	PDF	www.model.com/support/documentation/BOOK/sdk_um.pdf The Standard Developer's Kit is for use with Mentor Graphics QuickHDL.
Command Help	ASCII	type `help [command name]` at the prompt in the Main window
Error message help	ASCII	type `verror <msgNum>` at the Main window or shell prompt
Tcl Man Pages (Tcl manual)	HTML	select **Main window > Help > Tcl Man Pages**, or find *contents.htm* in *\modeltech\docs\tcl_help_html*
Technotes	HTML	select Technotes dropdown on www.model.com/support

Table of Contents

Syntax and conventions

Chapter contents

Documentation conventions

This manual uses the following conventions to define ModelSim 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[a]
{ }	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)

a. 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/mcarnes/simprims
```

or

```
vlog top.v -y "C:/Documents and Settings/mcarnes/simprims"
```

HDL and SystemC item names

VHDL, Verilog, and SystemC items are organized hierarchically. Each of the following items 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
- **SystemC**
  module instantiation

Item name syntax

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

```
[<datasetName><datasetSeparator>][<pathSeparator>][<hierarchicalPath>]<item
Name>[<elementSelection>]
```

where

`datasetName`
is the logical name of the WLF file in which the item 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. See *Chapter 9 - WLF files (datasets) and virtuals* 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 (UM-623) 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 item names. Normally, '/' is used for VHDL and '.' is used for Verilog, although other characters (except '\') may be specified via the PathSeparator (UM-624) variable in the *modelsim.ini* file. This character must be different than the datasetSeparator. Both '.' and '/' can be used for SystemC.

`hierarchicalPath`
is a set of hierarchical instance names separated by a path separator and ending in a path separator prior to the itemName. For example, */top/proc/clk*.

`itemName`
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" (CR-15) for important information about using square brackets in ModelSim commands.

Record field selection - A record field is specified using a period "." followed by the name of the field.

C++ class, structure, and union member selection - A class, structure, or union member is specified using the record field specification syntax, described just above.

SystemC class/structure/union member specification

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

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

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

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

Example

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

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

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

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

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

Legal names for viewing this signal are:

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

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

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

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

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

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

```
exa spot.value
```

Specifying names

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

A simple name does not contain any hierarchy. It is simply the name of an item (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 item (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 items 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** command (CR-166).

The current context in the current or specified dataset is used when accessing items 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** command (CR-166) to set the current context to a different instance.

Here are some examples of item names and what they specify:

Syntax	Description
clk	specifies the item *clk* in the current context
/top/clk	specifies the item *clk* in the top-level design unit.
/top/block1/u2/clk	specifies the item *clk*, two levels down from the top-level design unit
block1/u2/clk	specifies the item *clk*, two levels down from the current context
array_sig[4]	specifies an index of an array item
{array_sig(1 to 10)}	specifies a slice of an array item in VHDL or SystemC; see "Escaping brackets and spaces in array slices" (CR-15) for more information
{mysignal[31:0]}	specifies a slice of an array item in Verilog or SystemC; see "Escaping brackets and spaces in array slices" (CR-15) for more information
record_sig.field	specifies a field of a record, a C++ class or structure member, or a C++ base class

Escaping brackets and spaces in array slices

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

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

For complete details on Tcl syntax, see "Tcl command syntax" (UM-594).

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. See "Environment variables" (UM-613) 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. SystemC names are case sensitive.

Extended identifiers

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

```
{\ext ident!\ }    # Note that trailing space before closing brace is required

\\ext\ ident\!\\    # All non-alpha characters escaped
```

Wildcard characters

Wildcard characters can be used in HDL item names in some simulator commands. Conventions for wildcards are as follows:

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 (CR-172)

The *WildcardFilter* Tcl preference variable filters matching items for the add wave, add log, add list, and find commands.

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.

ModelSim variables

ModelSim variables can be referenced in simulator commands by preceding the name of the variable with the dollar sign ($) character. ModelSim uses global Tcl variables for simulator state variables, simulator control variables, simulator preference variables, and user-defined variables (see "Preference variables located in Tcl files" (UM-631) for more information).

See *Appendix A - ModelSim variables* in the User's Manual for more information on variables.

The **report** command (CR-238) 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. See UserTimeUnit (UM-626).

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.

Comments in argument files

Argument files may be loaded with the **-f <filename>** argument of the **vcom**, **vlog**, **sccom** and **vsim** commands. The **-f <filename>** argument specifies a file that contains more command line arguments.

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.

Also, program arguments can be placed 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.

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:

  ```
  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

The simulator command history may be reviewed, or commands may be reused, with these shortcuts at the ModelSim/VSIM prompt:

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

Numbering conventions

Numbers in ModelSim can be expressed in either VHDL or Verilog style. Two styles can be used for VHDL numbers, one for Verilog.

VHDL numbering conventions

The first of two VHDL number styles is:

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

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.

Examples

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

The second VHDL number style is:

```
base "value"
```

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

Examples

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

Verilog numbering conventions

Verilog numbers are expressed in the style:

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

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.

Examples

```
’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 items within the List and Wave windows (expressions may also be used through the Edit > Search menu in both windows). The commands that use the expression format are:

compare add (CR-100), **compare clock** (CR-105), **compare configure** (CR-107), **configure** (CR-129), **down** (CR-157), **examine** (CR-167), **left** (CR-185), **right** (CR-244), **searchlog** (CR-255), **up** (CR-282), **virtual function** (CR-327), and **virtual signal** (CR-339)

Expressions may be typed directly on the VSIM command line, or you can use the "The GUI Expression Builder" (UM-395).

Expression typing

GUI expressions are typed. The supported types consist of six scalar types and two array types.

Scalar types

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

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

SystemC scalar types that are supported are: bool, int, unsigned int, long, unsigned long, short, unsigned short, char, unsigned char, double, float, enumeration, and signal state. SystemC signal states can be '0' '1' 'X' and 'Z'.

Array types

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

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

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

The supported SystemC array types are sc_bv<w>, sc_lv<w>, sc_int<w>, sc_uint<w>, sc_bigint<w>, sc_biguint<w>, sc_fixed<...>, and sc_ufixed<...>.

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 curly 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

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

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

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

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

Signal attributes

<name>’event
<name>’rising
<name>’falling
<name>’delayed()
<name>’hasX

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

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

See "Examples" (CR-27) below for further details on ’delayed and ’hasX.

Operators

Operator	Description
&&	boolean and
||	boolean or
!	boolean not
==	equal
!=	not equal
===	exact equal
!==	exact not equal
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
not/NOT/~	unary bitwise inversion
and/AND/&	bitwise and
nand/NAND	bitwise nand
or/OR/|	bitwise or
nor/NOR	bitwise nor
xor/XOR	bitwise xor
xnor/XNOR	bitwise xnor

Operator	Description
sll/SLL	shift left logical
sla/SLA	shift left arithmetic
srl/SRL	shift right logical
sra/SRA	shift right arithmetic
ror/ROR	rotate right
rol/ROL	rotate left
+	arithmetic add
-	arithmetic subtract
*	arithmetic multiply
/	arithmetic divide
mod/MOD	arithmetic modulus
rem/REM	arithmetic remainder
|<vector_expr>	OR reduction
^<vector_expr>	XOR reduction

Note: Arithmetic operators use the std_logic_arith package.

Casting

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

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

```
clk'event && (/top/xyz == 16'hffae)
```
This expression evaluates to a boolean true when signal *clk* changes and signal */top/xyz* is equal to hex ffae; otherwise is false.

```
clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)
```
Evaluates to a boolean true when signal *clk* just changed from low to high and signal *mystate* is the enumeration reading and signal */top/u3/addr* is equal to the specified 32-bit hex constant; otherwise is false.

```
(/top/u3/addr and 32'hff000000) == 32'hac000000
```
Evaluates to a boolean true when the upper 8 bits of the 32-bit signal */top/u3/addr* equals hex ac.

```
/top/signalA'delayed(10ns)
```
This expression returns */top/signalA* delayed by 10 ns.

```
/top/signalA'delayed(10 ns) && /top/signalB
```
This expression takes the logical AND of a delayed */top/signalA* with */top/signalB*.

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

`((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode == writing)`
Evaluates to a boolean true when WLF file time is between 23 and 54 microseconds, *clk* just changed from low to high, and signal mode is enumeration writing.

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

Signal and subelement naming conventions

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

Examples in Verilog and VHDL syntax:

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

All of the above examples are valid for SystemC.

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 Signals 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

```
&{ "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 and SystemC class/structure/union 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 via 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.

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.

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

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

Commands

Chapter contents

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**).

Note: ModelSim commands are case sensitive. Type them as they are shown in this reference.

Command reference table

The following table provides a brief description of each ModelSim command. Command details, arguments, and examples can be found at the page numbers given in the Command name column.

Command name	Action
.main clear (CR-44)	clears the Main window transcript
.wave.tree interrupt (CR-45)	halts the drawing of waves in the Wave window
.wave.tree zoomfull (CR-46)	zooms the Wave window from time zero to the current simulation time
.wave.tree zoomin (CR-47)	zooms in the Wave window by the specified factor
.wave.tree zoomlast (CR-48)	zooms the Wave window to the previous setting
.wave.tree zoomout (CR-49)	zooms out the Wave window by the specified factor
.wave.tree zoomrange (CR-50)	zooms the Wave to the specified range
abort (CR-51)	halts the execution of a macro file interrupted by a breakpoint or error
add button (CR-52)	adds a user-defined button to the Main window button bar
add dataflow (CR-54)	adds the specified item to the Dataflow window
add list (CR-55)	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** (CR-187)
add_menu (CR-58)	adds a menu to the menu bar of the specified window, using the specified menu name
add_menucb (CR-60)	creates a checkbox within the specified menu of the specified window
add_menuitem (CR-61)	creates a menu item within the specified menu of the specified window
add_separator (CR-62)	adds a separator as the next item in the specified menu path in the specified window
add_submenu (CR-63)	creates a cascading submenu within the specified menu path of the specified window
add wave (CR-64)	adds VHDL signals and variables, and Verilog nets and registers to the Wave window
alias (CR-68)	creates a new Tcl procedure that evaluates the specified commands
assertion fail (CR-69)	configures fail tracking for PSL assertions
assertion pass (CR-71)	configures pass tracking for PSL assertions
assertion report (CR-73)	produces textual summary of PSL assertion results

Command name	Action
batch_mode (CR-75)	returns a 1 if ModelSim is operating in batch mode, otherwise returns a 0
bd (CR-76)	deletes a breakpoint
bookmark add wave (CR-77)	adds a bookmark to the specified Wave window
bookmark delete wave (CR-78)	deletes bookmarks from the specified Wave window
bookmark goto wave (CR-79)	zooms and scrolls a Wave window using the specified bookmark
bookmark list wave (CR-80)	displays a list of available bookmarks
bp (CR-81)	sets a breakpoint
cd (CR-84)	changes the ModelSim local directory to the specified directory
cdbg (CR-85)	provides command-line equivalents of the menu options that are available for "C Debug" (UM-473)
change (CR-87)	modifies the value of a VHDL variable or Verilog register variable
change_menu_cmd (CR-89)	changes the command to be executed for a specified menu item label, in the specified menu, in the specified window
check contention add (CR-90)	enables contention checking for the specified nodes
check contention config (CR-92)	writes checking messages to a file
check contention off (CR-93)	disables contention checking for the specified nodes
check float add (CR-94)	enables float checking for the specified nodes
check float config (CR-95)	writes checking messages to a file
check float off (CR-96)	disables float checking for the specified nodes
check stable off (CR-97)	disables stability checking
check stable on (CR-98)	enables stability checking on the entire design
checkpoint (CR-99)	saves the state of your simulation
compare add (CR-100)	compares signals in a reference design against signals in a test design
compare annotate (CR-104)	marks a compare difference as "ignore" or tags it with a text message
compare clock (CR-105)	defines a clock to be used with clocked-mode comparisons
compare configure (CR-107)	modifies options for compare signals or regions
compare continue (CR-109)	continues difference computation that had been suspended
compare delete (CR-110)	deletes a signal or region from the current comparison
compare end (CR-111)	closes the currently open comparison
compare info (CR-112)	lists the results of the comparison

Command name	Action
compare list (CR-113)	lists all the compare add commands currently in effect
compare options (CR-114)	sets defaults for options used in other compare commands
compare reload (CR-118)	reloads a comparison previously saved with the compare savediffs command
compare reset (CR-119)	clears the current compare differences
compare run (CR-120)	runs the comparison on selected signals
compare savediffs (CR-121)	saves comparison differences to a file that can be reloaded later
compare saverules (CR-122)	saves comparison setup information to a file that can be reloaded later
compare see (CR-123)	displays a comparison difference in the Wave window
compare start (CR-125)	starts a new dataset comparison
compare stop (CR-127)	halts active difference computation
compare update (CR-128)	updates the comparison differences
configure (CR-129)	invokes the List or Wave widget configure command for the current default List or Wave window
context (CR-133)	provides several operations on a context's name
coverage clear (CR-134)	clears all coverage data obtained during previous run commands
coverage exclude (CR-135)	loads an exclusion filter file
coverage reload (CR-136)	seeds the coverage statistics with the output of a previous **coverage report** command
coverage report (CR-137)	produces a textual output of the coverage statistics that have been gathered up to this point
coverage save (CR-140)	saves current coverage statistics to a file that can be reloaded later, preserving instance-specific information
dataset alias (CR-141)	assigns an additional name to a dataset
dataset clear (CR-142)	clears the current simulation WLF file
dataset close (CR-143)	closes a dataset
dataset info (CR-144)	reports information about the specified dataset
dataset list (CR-145)	lists the open dataset(s)
dataset open (CR-146)	opens a dataset and references it by a logical name
dataset rename (CR-147)	changes the logical name of an opened dataset
dataset save (CR-148)	saves data from the current WLF file to a specified file
dataset snapshot (CR-149)	saves data from the current WLF file at a specified interval

Command name	Action
delete (CR-151)	removes HDL items from either the List or Wave window
describe (CR-152)	displays information about the specified HDL item
disablebp (CR-153)	turns off breakpoints and when commands
disable_menu (CR-154)	disables the specified menu within the specified window
disable_menuitem (CR-155)	disables the specified menu item within the specified menu path of the specified window
do (CR-156)	executes commands contained in a macro file
down (CR-157)	searches for signal transitions or values in the specified List window
drivers (CR-159)	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 (CR-160)	dumps the contents of the *vsim.wlf* file in a readable format
echo (CR-161)	displays a specified message in the Main window
edit (CR-162)	invokes the editor specified by the EDITOR environment variable
enablebp (CR-163)	turns on breakpoints and when commands turned off by the **disablebp** command (CR-153)
enable_menu (CR-164)	enables a previously-disabled menu
enable_menuitem (CR-165)	enables a previously-disabled menu item
environment (CR-166)	displays or changes the current dataset and region environment
examine (CR-167)	examines one or more HDL items, and displays current values (or the values at a specified previous time) in the Main window
exit (CR-171)	exits the simulator and the ModelSim application
find (CR-172)	displays the full pathnames of all HDL items in the design whose names match the name specification you provide
force (CR-176)	applies stimulus to VHDL signals and Verilog nets and registers
gdb dir (CR-179)	sets the source directory for FLI/PLI/VPI C source code when using C Debug
getactivecursortime (CR-180)	gets the time of the active cursor in the Wave window
getactivemarkertime (CR-181)	gets the time of the active marker in the List window
help (CR-182)	displays in the Main window a brief description and syntax for the specified command
history (CR-183)	lists the commands executed during the current session
lecho (CR-184)	takes one or more Tcl lists as arguments and pretty-prints them to the Main window

Command name	Action
left (CR-185)	searches left (previous) for signal transitions or values in the specified Wave window
log (CR-187)	creates a wave log format (WLF) file containing simulation data for all HDL items whose names match the provided specifications
lshift (CR-189)	takes a Tcl list as argument and shifts it in-place one place to the left, eliminating the 0th element
lsublist (CR-190)	returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern
macro_option (CR-191)	controls the speed and delay of macro (DO file) playback, plus the level of debugging feedback
mem display (CR-192)	displays the memory contents of a selected instance to the screen
mem list (CR-194)	displays a flattened list of all memory instances in the current or specified context after a design has been elaborated
mem load (CR-195)	updates the simulation memory contents of a specified instance
mem save (CR-198)	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 (CR-200)	finds and prints to the screen the first occurring match of a specified memory pattern in the specified memory instance
modelsim (CR-202)	starts the ModelSim GUI without prompting you to load a design; valid only for Windows platforms
next (CR-203)	continues a search; see the **search** command (CR-253)
noforce (CR-204)	removes the effect of any active **force** (CR-176) commands on the selected HDL items
nolog (CR-205)	suspends writing of data to the WLF file for the specified signals
notepad (CR-207)	opens a simple text editor
noview (CR-208)	closes a window in the ModelSim GUI
nowhen (CR-209)	deactivates selected **when** (CR-375) commands
onbreak (CR-210)	specifies command(s) to be executed when running a macro that encounters a breakpoint in the source code
onElabError (CR-211)	specifies one or more commands to be executed when an error is encountered during elaboration
onerror (CR-212)	specifies one or more commands to be executed when a running macro encounters an error
pause (CR-213)	interrupts the execution of a macro

Command name	Action
play (CR-214)	plays a sequence of keyboard and mouse actions that were previously saved to a file with the **record** command (CR-237)
pop (CR-215)	moves one level up the C callstack
power add (CR-216)	specifies the signals or nets to track for power information
power report (CR-217)	writes out the power information for the specified signals or nets
power reset (CR-218)	resets power information to zero for the signals or nets specified with the **power add** command (CR-216)
precision (CR-219)	determines how real numbers display in the GUI
printenv (CR-220)	echoes to the Main window the current names and values of all environment variables
profile clear (CR-221)	clears any data that has been gathered during previous **run** commands
profile interval (CR-222)	selects the frequency with which the profiler collects samples during a run command
profile off (CR-223)	discontinues runtime profiling
profile on (CR-224)	enables runtime analysis of where your simulation is spending its time
profile option (CR-225)	allows various profiling options to be changed
profile report (CR-226)	produces a textual output of the profiling statistics that have been gathered up to this point
project (CR-227)	performs common operations on new projects
property list (CR-228)	changes one or more properties of the specified signal, net, or register in the List window (UM-286)
property wave (CR-229)	changes one or more properties of the specified signal, net, or register in the Wave window (UM-337)
push (CR-231)	moves one level down the C callstack
pwd (CR-232)	displays the current directory path in the Main window
quietly (CR-233)	turns off transcript echoing for the specified command
quit (CR-234)	exits the simulator
radix (CR-235)	specifies the default radix to be used
record (CR-237)	starts recording a replayable trace of all keyboard and mouse actions
report (CR-238)	displays the value of all simulator control variables, or the value of any simulator state variables relevant to the current simulation
restart (CR-240)	reloads the design elements and resets the simulation time to zero

Command name	Action
restore (CR-242)	restores the state of a simulation that was saved with a **checkpoint** command (CR-99) during the current invocation of vsim
resume (CR-243)	resumes execution of a macro file after a **pause** command (CR-213) or a breakpoint
right (CR-244)	searches right (next) for signal transitions or values in the specified Wave window
run (CR-246)	advances the simulation by the specified number of timesteps
sccom (CR-248)	compiles SystemC design units
scgenmod (CR-251)	creates a VHDL entity's or Verilog module's equivalent SystemC foreign module declaration, writing it to standard output
search (CR-253)	searches the specified window for one or more items matching the specified pattern(s)
searchlog (CR-255)	searches one or more of the currently open logfiles for a specified condition
seetime (CR-257)	scrolls the List or Wave window to make the specified time visible
setenv (CR-258)	sets an environment variable
shift (CR-259)	shifts macro parameter values down one place
show (CR-260)	lists HDL items and subregions visible from the current environment
simstats (CR-261)	reports performance-related statistics about active simulations
splitio (CR-262)	operates on a VHDL inout or out port to create a new signal having the same name as the port suffixed with "__o"
status (CR-263)	lists all currently interrupted macros
step (CR-264)	steps to the next HDL statement
stop (CR-265)	stops simulation in batch files; used with the **when** command (CR-375)
tb (CR-266)	displays a stack trace for the current process in the Main window
tcheck_set (CR-267)	modifies a timing check's reporting or X generation status
tcheck_status (CR-269)	prints to the Transcript the current status of timing checks
toggle add (CR-271)	enables collection of toggle statistics for the specified nodes
toggle disable (CR-273)	disables collection of toggle statistics for the specified nodes
toggle enable (CR-274)	re-enables collection of toggle statistics for the specified nodes
toggle report (CR-275)	displays to the screen a list of all nodes that have not transitioned to both 0 and 1 at least once
toggle reset (CR-276)	resets the toggle counts to zero for the specified nodes

Command name	Action
transcribe (CR-277)	displays a command in the Main window, then executes the command
transcript (CR-278)	controls echoing of commands executed in a macro file; also works at top level in batch mode
transcript file (CR-279)	sets or queries the pathname for the transcript file
tssi2mti (CR-280)	converts a vector file in Fluence Technology (formerly TSSI) Standard Events Format into a sequence of **force** (CR-176) and **run** (CR-246) commands
unsetenv (CR-281)	deletes an environment variable
up (CR-282)	searches for signal transitions or values in the specified List window
vcd add (CR-284)	adds the specified items to the VCD file
vcd checkpoint (CR-285)	dumps the current values of all VCD variables to the VCD file
vcd comment (CR-286)	inserts the specified comment in the VCD file
vcd dumpports (CR-287)	creates a VCD file that captures port driver data
vcd dumpportsall (CR-289)	creates a checkpoint in the VCD file that shows the current values of all selected ports
vcd dumpportsflush (CR-290)	flushes the VCD buffer to the VCD file
vcd dumpportslimit (CR-291)	specifies the maximum size of the VCD file
vcd dumpportsoff (CR-292)	turns off VCD dumping and records all dumped port values as x
vcd dumpportson (CR-293)	turns on VCD dumping and records the current values of all selected ports
vcd file (CR-294)	specifies the filename and state mapping for the VCD file created by a **vcd add** command (CR-284)
vcd files (CR-296)	specifies the filename and state mapping for the VCD file created by a **vcd add** command (CR-284); supports multiple VCD files
vcd flush (CR-298)	flushes the contents of the VCD file buffer to the VCD file
vcd limit (CR-299)	specifies the maximum size of the VCD file
vcd off (CR-300)	turns off VCD dumping and records all VCD variable values as x
vcd on (CR-301)	turns on VCD dumping and records the current values of all VCD variables
vcd2wlf (CR-302)	translates VCD files into WLF files
vcom (CR-303)	compiles VHDL design units
vcover convert (CR-310)	converts a 5.7 coverage file to a 5.8 format
vcover merge (CR-311)	merges multiple coverage data files
vcover stats (CR-313)	produces summary statistics from multiple coverage data files

Command name	Action
vdel (CR-315)	deletes a design unit from a specified library
vdir (CR-316)	lists the contents of a design library
verror (CR-317)	prints a detailed description of a message number
vgencomp (CR-318)	writes a Verilog module's equivalent VHDL component declaration to standard output
view (CR-320)	opens a ModelSim window and brings it to the front of the display
virtual count (CR-322)	counts the number of currently defined virtuals that were not read in using a macro file
virtual define (CR-323)	prints the definition of the virtual signal or function in the form of a command that can be used to re-create the object
virtual delete (CR-324)	removes the matching virtuals
virtual describe (CR-325)	prints a complete description of the data type of one or more virtual signals
virtual expand (CR-326)	produces a list of all the non-virtual objects contained in the virtual signal(s)
virtual function (CR-327)	creates a new signal that consists of logical operations on existing signals and simulation time
virtual hide (CR-330)	sets a flag in the specified real or virtual signals so that the signals do not appear in the Signals window
virtual log (CR-331)	causes the sim-mode dependent signals of the specified virtual signals to be logged by the simulator
virtual nohide (CR-333)	resets the flag set by a virtual hide command
virtual nolog (CR-334)	stops the logging of the specified virtual signals
virtual region (CR-336)	creates a new user-defined design hierarchy region
virtual save (CR-337)	saves the definitions of virtuals to a file
virtual show (CR-338)	lists the full path names of all the virtuals explicitly defined
virtual signal (CR-339)	creates a new signal that consists of concatenations of signals and subelements
virtual type (CR-342)	creates a new enumerated type
vlib (CR-344)	creates a design library
vlog (CR-345)	compiles Verilog design units
vmake (CR-355)	creates a makefile that can be used to reconstruct the specified library
vmap (CR-356)	defines a mapping between a logical library name and a directory
vsim (CR-357)	loads a new design into the simulator

Command name	Action
vsim<info> (CR-373)	returns information about the current vsim executable
vsource (CR-374)	specifies an alternative file to use for the current source file
when (CR-375)	instructs ModelSim to perform actions when the specified conditions are met
where (CR-380)	displays information about the system environment
wlf2log (CR-381)	translates a ModelSim WLF file(*vsim.wlf*) to a QuickSim II logfile
wlfman (CR-384)	outputs information about or a new WLF file from an existing WLF file
wlfrecover (CR-387)	attempts to repair an incomplete WLF file
write cell_report (CR-388)	creates a report of cell instances in the design that are optimized (-fast)
write format (CR-389)	records the names and display options in a file of the HDL items currently being displayed in the List or Wave window
write list (CR-391)	records the contents of the most recently opened or specified List window in a list output file
write preferences (CR-392)	saves the current GUI preference settings to a Tcl preference file
write report (CR-393)	prints a summary of the design being simulated
write transcript (CR-394)	writes the contents of the Main window transcript to the specified file
write tssi (CR-395)	records the contents of the default or specified List window in a “TSSI format” file
write wave (CR-397)	records the contents of the most currently opened or specified Wave window in PostScript format

.main clear

The **.main clear** command clears the transcript. The behavior is the same as the Main window **File > Transcript > Clear Transcript** menu selection.

Syntax

```
.main clear
```

Arguments

None.

See also

Main window (UM-262)

.wave.tree interrupt

The **.wave.tree interrupt** command halts the drawing of waves in the Wave window. This command can be useful when you have a large WLF file that is taking a long time to display.

Syntax

```
.wave.tree interrupt
```

Arguments

None.

.wave.tree zoomfull

The **.wave.tree zoomfull** command redraws the Wave window to show the entire simulation from time 0 to the current simulation time. The behavior is the same as the Wave window (UM-337) **View > Zoom > Zoom Full** menu selection.

Returns the zoom range as two time values.

Syntax

```
.wave.tree zoomfull
```

Arguments

None.

See also

.wave.tree zoomin (CR-47), **.wave.tree zoomlast** (CR-48), **.wave.tree zoomout** (CR-49), **.wave.tree zoomrange** (CR-50), "Zooming - changing the waveform display range" (UM-360)

Example

```
.wave.tree zoomfull
# {0 ns}{2310 ns}
```

.wave.tree zoomin

The **.wave.tree zoomin** command allows you to zoom in the Wave window by some factor. The behavior is similar to the Wave window (UM-337) **View > Zoom > Zoom In** menu selection.

Returns the zoom range as two time values.

Syntax

```
.wave.tree zoomin
  <factor>
```

Arguments

`<factor>`
A number that specifies how much you want to zoom in the Wave window. Required.

See also

.wave.tree zoomfull (CR-46), **.wave.tree zoomlast** (CR-48), **.wave.tree zoomout** (CR-49), **.wave.tree zoomrange** (CR-50)

Example

```
.wave.tree zoomin 2
# {577 ns}{1733 ns}
```

.wave.tree zoomlast

The **.wave.tree zoomlast** command zooms the Wave window to the setting prior to the most recent zoom change. The behavior is the same as the Wave window (UM-337) **View > Zoom > Zoom Last** menu selection.

Returns the zoom range as two time values.

Syntax

```
.wave.tree zoomlast
```

Arguments

None.

See also

.wave.tree zoomfull (CR-46), **.wave.tree zoomin** (CR-47), **.wave.tree zoomout** (CR-49), **.wave.tree zoomrange** (CR-50)

Example

```
.wave.tree zoomlast
# {0 ns}{2310 ns}
```

.wave.tree zoomout

The **.wave.tree zoomout** command allows you to zoom out the Wave window by some factor. The behavior is similar to the Wave window (UM-337) **View > Zoom > Zoom Out** menu selection.

Returns the zoom range as two time values.

Syntax

```
.wave.tree zoomout
  <factor>
```

Arguments

`<factor>`
A number that specifies how much you want to zoom out the Wave window. Required.

See also

.wave.tree zoomfull (CR-46), **.wave.tree zoomin** (CR-47), **.wave.tree zoomlast** (CR-48), **.wave.tree zoomrange** (CR-50)

Example

```
.wave.tee zoomout 2
# {865 ns}{1445 ns}
```

.wave.tree zoomrange

The **.wave.tree zoomrange** command lets you set the zoom range for the Wave window. The behavior is the same as the Wave window (UM-337) **View > Zoom > Zoom Range** menu selection.

Returns the zoom range as two time values.

Syntax

```
.wave.tree zoomrange
  [<time1> [<time2>]]
```

Arguments

`<time1>`

`<time2>`

time1 and time2 are floating point numbers that specify a zoom range. If neither number is specified, the command returns the current zoom range. If only time1 is specified, then the zoom range is set to start at 0 and end at time1.

Either range number may include an optional VHDL resolution time-unit. The resolution and range number must be enclosed in either quotes or curly brackets (see the example below). If not specified the resolution defaults to the UserTimeUnit (UM-626) set in the *modelsim.ini* file.

Examples

```
.wave.tree zoomrange {.5 us} {1.75 us}
# {500 ns} {1750 ns}
```

Zooms the Wave window between .5 us and 1.75 us and returns the zoom range in current simulator time units.

See also

.wave.tree zoomfull (CR-46), **.wave.tree zoomin** (CR-47), **.wave.tree zoomlast** (CR-48), **.wave.tree zoomout** (CR-49)

abort

The **abort** 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. The **abort** command may be used within a macro to return early.

Syntax

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

Arguments

`<n> | all`
An integer giving the number of nested macro levels to abort; **all** aborts all levels. Optional. Default is 1.

See also

onbreak (CR-210), **onElabError** (CR-211), **onerror** (CR-212)

add button

The **add button** command adds a user-defined button to the Main window button bar. New buttons are added to the right end of the bar. You can also add buttons with a ModelSim tool: "The Button Adder" (UM-400).

Returns the path name of the button widget created.

Syntax

```
add button
  <Text> <Cmd> [Disable | NoDisable] [{<option> <value> ...}]
```

Arguments

`<Text>`
The label to appear on the face of the button. Required.

`<Cmd>`
The command to be executed when the button is clicked with the left mouse button. To echo the command and display the return value in the Main window, prefix the command with the **transcribe** command (CR-277). **Transcribe** will also echo the results to the transcript window. Required.

`Disable | NoDisable`
If Disable, the button will be grayed-out during a run and not active. If NoDisable, the button will continue to be active during a run. Optional. The default is Disable.

`{<option> <value> ...}`
A list of option-value pairs that will be applied to the button widget. Optional. Any properties belonging to Tk button widgets may be set. Useful options are foreground color (**-fg**), background color (**-bg**), width (**-width**), and relief (**-relief**).

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

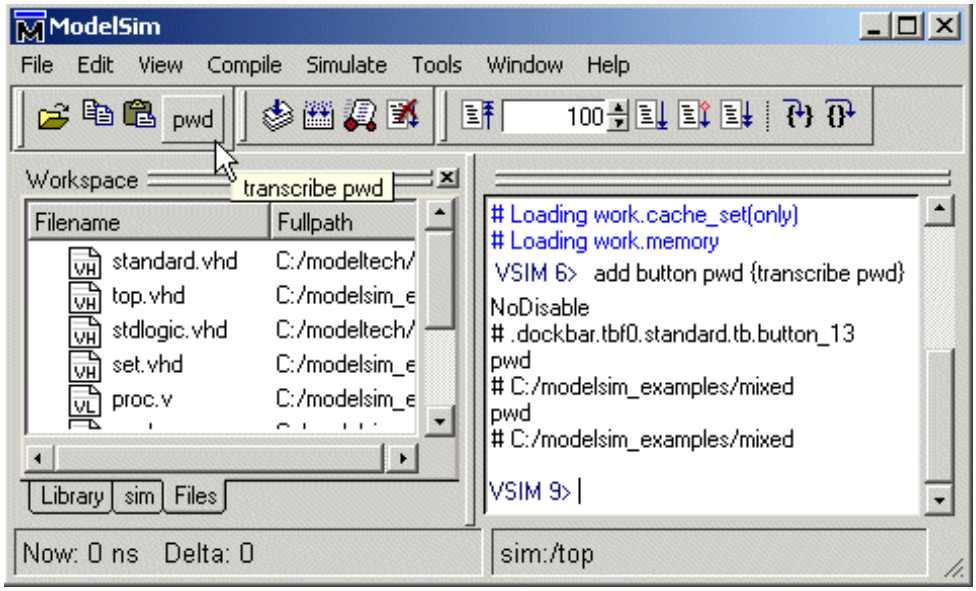
```
.dockbar.tbf0.standard.button_51 config
```

Note: Because the arguments are positional, a Disable | NoDisable option must be specified in order to use the options argument.

Examples

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

Creates a button labeled “pwd” that invokes the **transcribe** command (CR-277) with the **pwd** Tcl command, and echoes the command and its results to the Main window (see graphic below). The button remains active during a run.




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

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

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

Creates a “doit” button and underlines the second character of the label, the "o" of "doit".

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

Changes the button command to "run 10000" and changes the button background color to red.

See also

transcribe (CR-277), "The Button Adder" (UM-400) tool

add dataflow

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

Syntax

```
add dataflow
  <item> [-window <wname>]
```

`<item>`
Specifies a process, signal, net, or register that you want to add to the Dataflow window. Required. Multiple items separated by spaces may be specified. Wildcards are allowed. (Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching items with wildcard patterns.)

`-window <wname>`
Adds the items to the specified Dataflow window <wname> (e.g., dataflow2). Optional. Used to specify a particular window when multiple instances of that window type exist. Selects an existing window; does not create a new window. Use the **view** command (CR-320) with the **-new** option to create a new window.

See also

Dataflow window (UM-270)

add list

The **add list** command adds the following items and their values to the List window: VHDL signals and variables; Verilog nets and registers; and SystemC primitive channels (signals). User-defined buses may also be added.

If no port mode is specified, **add list** will display all items in the selected region with names matching the item name specification.

Limitations: VHDL variables and Verilog memories can be listed using the variable's full name only (no wildcards).

Syntax

```
add list
  [-allowconstants] [-depth <level>] [-in] [-inout] [-internal]
  [[<item_name> | {<item_name> {sig1 sig2 sig3 ...}}] ...] ...
  [-label <name>] [-nodelta] [-notrigger | -trigger] [-optcells] [-out]
  [-ports] [-<radix>] [-recursive] [-width <n>] [-window <wname>]
```

Arguments

`-allowconstants`
For use with wildcard searches. Specifies that constants matching the wildcard search should be added to the List window. Optional. By default, constants are ignored because they do not change.

`-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.

`-in`
For use with wildcard searches. Specifies that the scope of the search is to include ports of mode IN if they match the **item_name** specification. Optional.

`-inout`
For use with wildcard searches. Specifies that the scope of the search is to include ports of mode INOUT if they match the **item_name** specification. Optional.

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

`<item_name>`
Specifies the name of the item to be listed. Optional. Wildcard characters are allowed. (Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching items with wildcard patterns.) Variables may be added if preceded by the process name. For example,

```
add list myproc/int1
```

`{<item_name> {sig1 sig2 sig3 ...}}`
Creates a user-defined bus in place of **item_name**; 'sig*i*' are signals to be concatenated within the user-defined bus. Optional. Specified items may be either scalars or various sized arrays as long as they have the same element enumeration type.

`-label <name>`
Specifies an alternative signal name to be displayed as a column heading in the listing. Optional. This alternative name is not valid in a **force** (CR-176) or **examine** (CR-167) command; however, it can be used in a **search** command (CR-253) with the **-list** option.

`-nodelta`
Specifies that the delta column not be displayed when adding signals to the List window. Optional. Identical to **configure list -delta none**.

`-notrigger`
Specifies that items are to be listed, but does not cause the List window to be updated when the items change value. Optional.

`-optcells`
Makes Verilog optimized cell ports visible when using wildcards. Optional. By default Verilog optimized cell ports are not selected even if they match the specified wildcard pattern.

`-out`
For use with wildcard searches. Specifies that the scope of the search is to include ports of mode OUT if they match the **item_name** specification. Optional.

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

`-<radix>`
Specifies the radix for the items that follow in the command. Optional. Valid entries (or any unique abbreviations) are: binary, ascii, unsigned, decimal, octal, hex, symbolic, 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 (CR-235). You can change the default radix permanently by editing the DefaultRadix (UM-623) variable in the *modelsim.ini* file.

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

`-recursive`
For use with wildcard searches. 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.

`-trigger`
Specifies that items are to be listed and causes the List window to be updated when the items change value. Optional. Default.

`-width <n>`
Specifies the column width in characters. Optional.

`-window <wname>`
Adds items to the specified List window <wname> (e.g., list2). Optional. Used to specify a particular window when multiple instances of that window type exist. Selects an existing window; does not create a new window. Use the **view** command (CR-320) with the **-new** option to create a new window.

Examples

```
add list -r /*
```
Lists all items in the design.

```
add list *
```
Lists all items in the region.

```
add list -in *
```
Lists all input ports in the region.

```
add list a -label sig /top/lower/sig {array_sig(9 to 23)}
```
Displays a List window containing three columns headed *a*, *sig*, and *array_sig(9 to 23)*.

```
add list clk -notrigger a b c d
```
Lists *clk*, *a*, *b*, *c*, and *d* only when *clk* changes.

```
config list -strobeperiod {100 ns} -strobestart {0 ns} -usestrobe 1
add list -notrigger clk a b c d
```
Lists *clk*, *a*, *b*, *c*, and *d* every 100 ns.

```
add list -hex {mybus {msb {opcode(8 downto 1)} data}}
```
Creates a user-defined bus named "mybus" consisting of three signals; the bus is displayed in hex.

```
add list vec1 -hex vec2 -dec vec3 vec4
```
Lists the item *vec1* using symbolic values, lists *vec2* in hexadecimal, and lists *vec3* and *vec4* in decimal.

See also

add wave (CR-64), **log** (CR-187), "Extended identifiers" (CR-16)

add_menu

The **add_menu** command adds a menu to the menu bar of the specified window, using the specified menu name. Use the **add_menuitem** (CR-61), **add_separator** (CR-62), **add_menucb** (CR-60), and **add_submenu** (CR-63) commands to complete the menu.

Returns the full Tk pathname of the new menu.

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

Syntax

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

Arguments

`<window_name>`
Tk path of the window to contain the menu. Required.

Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

`<menu_name>`
Name to be given to the Tk menu widget. Required.

`<shortcut>`
Number of the letter in the menu name that is to be used as the shortcut. Numbering starts with 0 (i.e., first letter = 0, second letter = 1, third letter = 2, etc.). Optional unless you specify **-hide_menubutton**, in which case **<shortcut>** is required. Default is "-1", which indicates no shortcut is to be used.

`-hide_menubutton`
Causes the new menu not to be displayed. Optional. You can add the menu later by calling **tk_popup** on the menu path widget. Note that you must specify **<shortcut>** if you specify **-hide_menubutton**.

Examples

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

```
view signals
set myglobalvar(.signals:one) 0
set myglobalvar(.signals:two) 1
proc AddMyMenus {wname} {

    global myglobalvar
    set cmd1 "echo my_own_thing $wname"
    set cmd2 "echo my_to_upper  $wname"
    set cmd3 "echo my_to_lower  $wname"

    #              WindowName  Menu   MenuItem label        Command
    #              ----------  ----   --------------------  -------
    add_menu       $wname      mine   0;# 0th letter (M) is underlined
    add_menuitem   $wname      mine   "Do My Own Thing..."  $cmd1
    add_separator  $wname      mine   ;#---------------------------
    add_submenu    $wname      mine   changeCase
    add_menuitem   $wname      mine.changeCase "To Upper"   $cmd2
    add_menuitem   $wname      mine.changeCase "To Lower"   $cmd3
    add_submenu    $wname      mine   vars
    add_menucb     $wname      mine.vars "Feature One"      -variable
                                                  myglobalvar($wname:one)
                                  -onvalue 1 -offvalue 0 -indicatoron 1
}
AddMyMenus .signals
```

This example is available in the following DO file: *<install_dir>/modeltech/examples/addmenu.do*. You can run the DO file to add the "Mine" menu shown in the illustration, or modify the file for different results.

To execute the DO file, select **Tools > Execute Macro** (Main window), or use the **do** command (CR-156).

See also

add_menucb (CR-60), **add_menuitem** (CR-61), **add_separator** (CR-62), **add_submenu** (CR-63), **change_menu_cmd** (CR-89)

add_menucb

The **add_menucb** command creates a checkbox within the specified menu of the specified window. A checkbox is a small box with a label. Clicking on the box will toggle the state, from on to off or the reverse. When the box is "on", the Tcl global variable <var> is set to <onval>. When the box is "off", the global variable is set to <offval>. Also, if something else changes the global variable, its current state is reflected in the state of the checkbox. Returns nothing.

Syntax

```
add_menucb
  <window_name> <menu_name> <Text> -variable <var> -onvalue <onval>
  -offvalue <offval> [-indicatoron <val>]
```

Arguments

`<window_name>`
Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).

`<menu_name>`
Name of the Tk menu widget. Required.

`<Text>`
Text to be displayed next to the checkbox. Required.

`-variable <var>`
Global Tcl variable to be reflected and changed. Required.

`-onvalue <onval>`
Value to set the global Tcl variable to when the box is "on". Required.

`-offvalue <offval>`
Value to set the global Tcl variable to when the box is "off". Required.

`-indicatoron <val>`
0 or 1. If 1, the status indicator is displayed. Otherwise it is not displayed. Optional. The default is 1.

Examples

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

See also

add_menu (CR-58), **add_menuitem** (CR-61), **add_separator** (CR-62), **add_submenu** (CR-63), **change_menu_cmd** (CR-89)

The **add_menucb** command is also used as part of the **add_menu** (CR-58) example.

add_menuitem

The **add_menuitem** command creates a menu item within the specified menu of the specified window. May be used within a submenu. Returns nothing.

Syntax

```
add_menuitem
  <window_name> <menu_path> <Text> <Cmd> [<shortcut>]
```

Arguments

`<window_name>`
Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).

`<menu_path>`
Name of the Tk menu widget plus submenu path. Required.

`<Text>`
Text to be displayed. Required.

`<Cmd>`
The command to be executed when the menu item is selected with the left mouse button. To echo the command and display the return value in the Main window, prefix the command with the **transcribe** command (CR-277). **Transcribe** will also echo the results to the transcript window. Required.

`<shortcut>`
Number of the letter in the menu name that is to be used as the shortcut. Numbering starts with 0 (i.e., first letter = 0, second letter = 1, third letter = 2, etc.). Optional. Default is "-1", which indicates no shortcut is to be used.

Examples

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

See also

add_menu (CR-58), **add_menucb** (CR-60), **add_separator** (CR-62), **add_submenu** (CR-63), **change_menu_cmd** (CR-89)

The **add_menuitem** command is also used as part of the **add_menu** (CR-58) example.

add_separator

The **add_separator** command adds a separator as the next item in the specified menu path in the specified window. Returns nothing.

Syntax

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

Arguments

`<window_name>`
Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).

`<menu_path>`
Name of the Tk menu widget plus submenu path. Required.

Examples

```
add_separator $wname user
```

See also

add_menu (CR-58), **add_menucb** (CR-60), **add_menuitem** (CR-61), **add_submenu** (CR-63), **change_menu_cmd** (CR-89)

The **add_separator** command is also used as part of the **add_menu** (CR-58) example.

add_submenu

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

Returns the full Tk path to the new submenu widget.

Syntax

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

Arguments

`<window_name>`
Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).

`<menu_path>`
Name of the Tk menu widget plus submenu path. Required.

`<name>`
Name to be displayed on the submenu. Required.

`<shortcut>`
Number of the letter in the menu name that is to be used as the shortcut. Numbering starts with 0 (i.e., first letter = 0, second letter = 1, third letter = 2, etc.). Optional. Default is "-1", which indicates no shortcut is to be used.

See also

add_menu (CR-58), **add_menucb** (CR-60), **add_menuitem** (CR-61), **add_separator** (CR-62), **change_menu_cmd** (CR-89)

The **add_submenu** command is also used as part of the **add_menu** (CR-58) example.

add wave

The **add wave** command adds the following items to the List window: VHDL signals and variables; Verilog nets and registers; and SystemC primitive channels (signals). User-defined buses may also be added.

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

Limitations: VHDL variables and Verilog memories can be added using the variable's full name only (no wildcards).

Syntax

```
add wave
  [-allowconstants] [-color <standard_color_name>] [-depth <level>] [-expand
  <signal_name>] [-<format>] [-height <pixels>] [-in] [-inout] [-internal]
  [[-divider <divider_name>...] | [<item_name> | {<item_name> {sig1 sig2 sig3
  ...}}] ...] [-label <name>] [-noupdate] [-offset <offset>] [-optcells]
  [-out] [-ports] [-<radix>] [-recursive] [-scale <scale>] [-window <wname>]
```

Arguments

`-allowconstants`
For use with wildcard searches. Specifies that constants matching the wildcard search should be added to the Wave window. Optional. By default, constants are ignored because they do not change.

`-color <standard_color_name>`
Specifies the color used to display a waveform. Optional. These are the standard X Window color names, or rgb value (e.g., #357f77); enclose 2-word names ("light blue") in quotes.

`-depth <level>`
Restricts a recursive search (specified with the **-recursive** option) 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.

`-divider <divider_name>`
Adds a divider with the specified name. Optional. You can specify one or more names. All names listed after **-divider** are taken to be names.

`-expand <signal_name>`
Causes a compound signal to be expanded immediately, but only one level down. Optional. The <signal_name> is required, and may include wildcards.

`-<format>`
Specifies the display format of the items:

```
literal
logic
analog-step
analog-interpolated
analog-backstep
```

Optional. Literal waveforms are displayed as a box containing the item value. Logic signals may be U, X, 0, 1, Z, W, L, H, or '-'.

The way each state is displayed is specified by the logic type display preference (see "Preference variables located in INI files" (UM-617)). Analog signals are sized by **-scale** and by **-offset**. Analog-step changes to the new time before plotting the new Y. Analog-interpolated draws a diagonal line. Analog-backstep plots the new Y before moving to the new time. See "Editing and formatting items in the Wave window" (UM-347) for more information.

`-height <pixels>`
Specifies the height (in pixels) of the waveform. Optional.

`-in`
For use with wildcard searches. Specifies that the scope of the search is to include ports of mode IN if they match the item_name specification. Optional.

`-inout`
For use with wildcard searches. Specifies that the scope of the search is to include ports of mode INOUT if they match the item_name specification. Optional.

`-internal`
For use with wildcard searches. Specifies that the scope of the search is to include internal items (non-port items) if they match the item_name specification. Optional.

`<item_name>`
Specifies the names of items to be included in the Wave window display. Optional. Wildcard characters are allowed. Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching items with wildcard patterns. Variables may be added if preceded by the process name. For example,

```
add wave myproc/int1
```

`{<item_name> {sig1 sig2 sig3 ...}}`
Creates a user-defined bus with the name <item_name>; 'sig*i*' are signals to be concatenated within the user-defined bus. Optional.

Note: You can also select **Tools > Combine Signals** (Wave window) to create a user-defined bus.

`-label <name>`
Specifies an alternative name for the signal being added to the Wave window. Optional. For example,

```
add wave -label c clock
```

adds the *clock* signal, labeled as "c", to the Wave window.

This alternative name is not valid in a **force** (CR-176) or **examine** (CR-167) command; however, it can be used in a **search** command (CR-253) with the **wave** option.

`-noupdate`
Prevents the Wave window from updating when a series of **add wave** commands are executed in series. Optional.

`-offset <offset>`
Modifies an analog waveform's position on the display. Optional. The offset value is part of the wave positioning equation (see **-scale** below).

`-optcells`
Makes Verilog optimized cell ports visible when using wildcards. Optional. By default Verilog optimized cell ports are not selected even if they match the specified wildcard pattern.

`-out`
For use with wildcard searches. Specifies that the scope of the search is to include ports of mode OUT if they match the item_name specification. Optional.

`-ports`
For use with wildcard searches. Specifies that the scope of the listing is to include ports of modes IN, OUT, or INOUT. Optional.

`-<radix>`
Specifies the radix for the items that follow in the command. Optional.

Valid entries (or any unique abbreviations) are: binary, ascii character, unsigned decimal, octal, hex, symbolic, 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 (CR-235). You can change the default radix permanently by editing the DefaultRadix (UM-623) variable in the *modelsim.ini* file.

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

`-recursive`
For use with wildcard searches. 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.

`-scale <scale>`
Scales analog waveforms. Optional. The scale value is part of the wave positioning equation shown below.

The position and size of the waveform is given by:

```
(signal_value + <offset>) * <scale>
```

If *signal_value* + <*offset*> = *0*, the waveform will be aligned with its name. The <*scale*> value determines the height of the waveform, 0 being a flat line.

`-window <wname>`
Adds items to the specified window <wname> (e.g., wave2). Optional. Used to specify a particular window when multiple instances of that window type exist. Selects an existing window; does not create a new window. Use the **view** command (CR-320) with the **-new** option to create a new window.

Examples

```
add wave -logic -color gold out2
```
Displays an item named *out2*. The item is specified as being a logic item presented in gold.

```
add wave -hex {address {a_7 a_6 a_5 a_4 a_3 a_2 a_1 a_0}}
```
Displays a user-defined, hex formatted bus named *address*.

`add wave *`
Waves all items in the region.

`add wave -in *`
Waves all input ports in the region.

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

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 wave vec1 -hex vec2 -dec vec3 vec4`
Adds the item *vec1* to the Wave window using symbolic values, adds *vec2* in hexadecimal, and adds *vec3* and *vec4* in decimal.

See also

add list (CR-55), **log** (CR-187), "Extended identifiers" (CR-16), "Concatenation directives" (CR-29)

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 ModelSim commands (e.g., run, env, etc.) cannot be aliased.

Syntax

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

Arguments

`<name>`
Specifies the new procedure name to be used when invoking the commands.

`"<cmds>"`
Specifies the command or commands to be evaluated when the alias is invoked.

Examples

`alias`
Lists all aliases currently defined.

`alias <name>`
Lists the alias definition for the specified name if one exists.

```
alias myquit "write list ./mylist.save; quit -f"
```
Creates a Tcl procedure, "myquit", that when executed, writes the contents of the List window to the file *mylist.save* by invoking **write list** (CR-391), and quits ModelSim by invoking **quit** (CR-234).

assertion fail

The **assertion fail** command configures simulator behavior in response to an assertion failure.

Syntax

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

Arguments

`-action continue|break|exit`
Specify the action to take when an assertion fails. This option may be specified multiple times; it applies to all paths that follow it in the command line. One of the following values is required:

continue–No action taken. This is not the same as disabling an assertion since logging may still be enabled for the directive. This is the default value.

break–Halt simulation and return to the ModelSim prompt.

exit–Halt simulation and exit ModelSim.

You can change the permanent default by setting the AssertionFailAction (UM-621) variable in the *modelsim.ini* file.

`-disable`
Turns off failure tracking for the specified assertions. Optional. Assertion failure tracking is enabled by default. You can change the permanent default by setting the AssertionFailEnable (UM-621) variable in the *modelsim.ini* file.

`-enable`
Turns on failure tracking for the specified assertions. Optional. Default. You can change the permanent default by setting the AssertionFailEnable (UM-621) variable in the *modelsim.ini* file.

`-limit <count>|none`
Sets a limit on the number of times ModelSim responds to an assertion failing. Optional. By default the limit is set to 1. One of the following values is required:

<*count*>–Specify a whole number.

none–No limit; failure tracking remains enabled for the duration of the simulation.

Once the limit is reached for a particular assertion, ModelSim disables failure tracking on that assertion. ModelSim continues to respond to others if their limit has not been reached. You can change the permanent default by setting the AssertionFailLimit (UM-621) variable in the *modelsim.ini* file.

`-log on|off`
Specify whether to write a transcript message when an assertion fails. This option may be specified multiple times; it applies to all paths that follow it in the command line. One of the following values is required:

on–Enable transcript logging. Default.

off–Disable transcript logging.

You can change the permanent default by setting the AssertionFailLog (UM-621) variable in the *modelsim.ini* file.

`-recursive`
For use with wildcard matching. Specifies that the scope of the matching is to descend recursively into subregions. Optional. If omitted, the search is limited to the selected region. Applies to all paths specified in the command.

`<path>`
Specifies the assertions to be affected. Required. Multiple paths and wildcards are allowed. The path specifies assertions or a design region containing multiple assertions.

Examples

```
assertion fail -disable a.b.c.assert__0
```
Disables assertion *a.b.c.assert__0*.

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

```
assertion fail -log off a.b.c.assert__0 -log on a.b.c.assert__1
```
Disables logging for assertion *a.b.c.assert__0* but enables it for *a.b.c.assert__1*.

```
assertion fail -limit 4
```
Sets the failure response limit to 4. Each assertion failure will be responded to a maximum of 4 times during the current simulation.

See also

"Enabling/disabling failure and pass checking" (UM-510), "Setting failure and pass limits" (UM-512), "Setting failure action" (UM-513), **assertion pass** command (CR-71), and **assertion report** command (CR-73)

assertion pass

The **assertion pass** command configures simulator behavior in response to an assertion pass.

Syntax

```
assertion pass
  [-disable] [-enable] [-limit <count>|none] [-log on|off] [-recursive]
  <path> [<path>...]
```

Arguments

`-disable`
Turns off pass tracking for the specified assertions. Optional. Default. You can change the permanent default by setting the AssertionPassEnable (UM-621) variable in the *modelsim.ini* file.

`-enable`
Turns on pass tracking for the specified assertions. Optional. Assertion pass tracking is disabled by default. You can change the permanent default by setting the AssertionPassEnable (UM-621) variable in the *modelsim.ini* file.

`-limit <count>|none`
Sets a limit on the number of times ModelSim responds to an assertion pass. Optional. By default the limit is set to 1. One of the following values is required:

<*count*>–Specify a whole number.

none–No limit; pass tracking remains enabled for the duration of the simulation.

This limit is global; it is applied to each assertion in the simulation. Once the limit is reached for a particular assertion, ModelSim disables pass tracking on that assertion. ModelSim continues to respond to others if their limit has not been reached. You can change the permanent default by setting the AssertionPassLimit (UM-621) variable in the *modelsim.ini* file.

`-log on|off`
Specify whether to write a transcript message when an assertion passes. This option may be specified multiple times; it applies to all paths that follow it in the command line. One of the following values is required:

on–Enable transcript logging. Default.

off–Disable transcript logging.

You can change the permanent default by setting the AssertionPassLog (UM-621) variable in the *modelsim.ini* file.

`-recursive`
For use with wildcard matching. Specifies that the scope of the matching is to descend recursively into subregions. Optional. If omitted, the search is limited to the selected region. Applies to all paths specified in the command.

`<path>`
Specifies the assertions to be affected. Required. Multiple paths and wildcards are allowed. The path specifies assertions or a design region containing multiple assertions.

Examples

`assertion pass -enable -log on a.b.c.assert__0 -log off a.b.c.assert_1`
Enables assertions *a.b.c.assert__0* and *a.b.c.assert__1* and turns logging on for *a.b.c.assert__0* but not *a.b.c.assert__1*.

See also

"Enabling/disabling failure and pass checking" (UM-510), "Setting failure and pass limits" (UM-512), **assertion fail** command (CR-69), and **assertion report** command (CR-73)

assertion report

The **assertion report** command returns a status report for each assertion matching the path specification. By default the command prints a concise report containing only assertion names and their fail and pass counts. Adding the **-verbose** argument to the command will print the following:

- source language (PSL or other)
- assertion name (full path)
- design unit where the assertion is declared
- source language (VHDL)
- filename (line number)
- fail enable status (enabled or disabled)
- pass enable status (enabled or disabled)
- fail count
- pass count
- attempted flag (indicates whether the assertion has ever attempted evaluation)
- fail action
- fail log
- pass log
- fail limit
- pass limit

Normally, the report is formatted for users with one line of the report reserved for each assertion specified by the path(s).

For a more interactive look at this data, open the Assertion Browser in the ModelSim GUI. See "Viewing assertions in the Assertion Browser" (UM-507) for details.

Syntax

```
assertion report
  [-number] [-recursive] [-tcl_list] [-verbose] <path> [<path>...]
```

Arguments

`-number`
Report the number of assertions that match the path argument(s). This option overrides the normal report.

`-recursive`
For use with wildcard matching. Specifies that the scope of the matching is to descend recursively into subregions. Optional; if omitted, the search is limited to the selected region.

`-tcl_list`
Format the report as a Tcl list.

`-verbose`
Produces the detailed report as noted in the command description above. Optional.

`<path>`
Specifies the assertions on which to report. Required. Multiple names and wildcards are allowed. The path specifies assertions or a design region containing multiple assertions.

Example

```
VSIM 1> assertion report *
# ---------------------------------------------------------------------
# Name                            File(Line)          Failure Pass
#                                                     Count   Count
# ---------------------------------------------------------------------
# /tb/assert__reset_state         dramcon_sim.vhd(53)       0     0
# /tb/assert__test_read_response  dramcon_sim.vhd(59)       0     0
# /tb/assert__test_write_response dramcon_sim.vhd(60)       0     0
# /tb/assert__check_as_deasserts  dramcon_sim.vhd(64)       0     0
```

See also

"Viewing assertions in the Assertion Browser" (UM-507)

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

"ModelSim modes of operation" (UM-23)

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#>
```

Arguments

`<filename>`
Specifies the name of the source file in which the breakpoint is to be deleted. Required if an id# is not specified. The filename must match the one used previously to set the breakpoint, including whether a full pathname or a relative name was used.

`<line_number>`
Specifies the line number of the breakpoint to be deleted. Required if an id# is not specified.

`<id#>`
Specifies the id number of the breakpoint to be deleted. Required if a filename and line number are not specified. If you are deleting a C breakpoint, the id# will have a "c" prefix.

Examples

`bd alu.vhd 127`
Deletes the breakpoint at line 127 in the source file named *alu.vhd*.

`bd 5`
Deletes the breakpoint with id# 5.

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

`bd c.4`
Deletes the C breakpoint with id# c.4.

See also

bp (CR-81), **onbreak** (CR-210), *Chapter 14 - C Debug*

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 (see **write format** command (CR-389)).

Syntax

```
bookmark add wave
  <label> <zoomrange> <topindex> [-window <window_name>]
```

Arguments

`<label>`
Specifies the name for the bookmark. Required.

`<zoomrange>`
Specifies a list of two times with optional units. Required. These two times must be enclosed in braces ({}) or quotation marks ("").

`<topindex>`
Specifies the vertical scroll position of the window. Required. The number identifies which item the window should be scrolled to. For example, specifying 20 means the Wave window will be scrolled down to show the 20th item.

`-window <window_name>`
Specifies the window to which the bookmark will be added. Optional. If this argument is omitted, the bookmark is added in the current default Wave window.

Examples

```
bookmark add wave foo {{10 ns} {1000 ns}} 20
```
Adds 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 item in the window.

See also

bookmark delete wave (CR-78), **bookmark goto wave** (CR-79), **bookmark list wave** (CR-80), **write format** (CR-389)

bookmark delete wave

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

Syntax

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

Arguments

`<label>`
Specifies the name of the bookmark to delete. Required unless the -all switch is used.

`-all`
Specifies that all bookmarks in the window be deleted. Optional.

`-window <window_name>`
Specifies the window from which bookmark(s) will be deleted. Optional. If this argument is omitted, bookmark(s) in the current default Wave window are deleted.

Examples

`bookmark delete wave foo`
Deletes the bookmark named "foo" from the current default Wave window.

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

See also

bookmark add wave (CR-77), **bookmark goto wave** (CR-79), **bookmark list wave** (CR-80), **write format** (CR-389)

bookmark goto wave

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

Syntax

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

Arguments

`<label>`
Specifies the bookmark to go to. Required.

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

See also

bookmark add wave (CR-77), **bookmark delete wave** (CR-78), **bookmark list wave** (CR-80), **write format** (CR-389)

bookmark list wave

The **bookmark list wave** command displays a list of available bookmarks in the Main window transcript.

Syntax

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

Arguments

`-window <window_name>`
Specifies the Wave window for which you want a list of bookmarks. Optional. If this argument is omitted, ModelSim lists the bookmarks for the current default Wave window.

See also

bookmark add wave (CR-77), **bookmark delete wave** (CR-78), **bookmark goto wave** (CR-79), **write format** (CR-389)

bp

The **bp** or breakpoint command either sets a file-line breakpoint or returns a list of currently set breakpoints. A set breakpoint affects every instance in the design unless the -inst <region> argument is used.

Syntax

```
bp
  <filename> <line_number>
  [-c [<function_name> | <file_name>:<line#> | <line#> | *0x<hex_address>]]
  [-id <id#>] [-inst <region>] [-disable] [-cond {<condition_expression>}]
  [{<command>...}] | [-query <filename> [<line_number> [<line_number>]]]
```

Arguments

`<filename>`
Specifies the name of the source file in which to set the breakpoint. Required if you are setting HDL breakpoints.

`<line_number>`
Specifies the line number at which the breakpoint is to be set. Required if you are setting HDL breakpoints.

`-c [<function_name> | <file_name>:<line#> | <line#> | *0x<hex_address>]`
Sets a C breakpoint in SystemC designs, or when you are using "C Debug" (UM-473). The -c argument is required when setting C breakpoints to distinguish them from HDL breakpoints. See examples below.

`-id <id#>`
Attempts to assign this id number to the breakpoint. Optional. If the id number you specify is already used, ModelSim will return an error.

Note: Ids for breakpoints are assigned from the same pool as those used for the **when** command (CR-375). So, even if you haven't used an id number for a breakpoint, it's possible it is used for a **when** command.

`-inst <region>`
Sets the breakpoint so it applies only to the specified region. Optional.

`-disable`
Sets the breakpoint to a disabled state. Optional. You can enable the breakpoint later using the **enablebp** command (CR-163). By default, breakpoints are enabled when they are set.

`-cond {<condition_expression>}`
Specifies condition(s) that determine whether the breakpoint is hit. Optional. If the condition is true, the simulation stops at the breakpoint. If false, the simulation bypasses the breakpoint.

The condition can be an expression with these operators:

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

The operands may be item 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.

`{<command>...}`

Specifies one or more commands that are to be executed at the breakpoint. Optional. Multiple commands must be separated by semicolons (;) or placed on multiple lines. The entire command must be placed in curly braces.

Any commands that follow a **run** (CR-246) or **step** (CR-264) command will be ignored. A **run** or **step** command terminates the breakpoint sequence. This applies if macros are used within the **bp** command string as well. A **restore** (CR-242) command should not be used.

If many commands are needed after the breakpoint, they can be placed in a macro file.

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

Returns information about the breakpoints set in the specified file. The information returned varies depending on which arguments you specify. See the examples below for details.

Examples

`bp`
Lists 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 alu.vhd 147`
Sets a breakpoint in the source file *alu.vhd* at line 147.

`bp alu.vhd 147 {do macro.do}`
Executes the *macro.do* macro file after the breakpoint.

`bp -disable test.vhd 22 {echo [exa var1]; echo [exa var2]}`
Sets 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 (CR-163).

`bp test.vhd 14 {if {$now /= 100} then {cont}}`
Sets a breakpoint in every instantiation of the file *test.vhd* at line 14. When that breakpoint is executed, the command is run. This command causes the simulator to continue if the current simulation time is not 100.

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

`bp -query testadd.vhd 48`
Lists details about the breakpoint on line 48. The output comprises six pieces of information: the first item (0 or 1) designates whether a breakpoint exists on the line (1 = exists, 0 = doesn't exist); the second item is always 1; the third item is the file name in the compiled source; the fourth item is the breakpoint line number; the fifth item is the breakpoint id; and the sixth item (0 or 1) designates whether the breakpoint is enabled (1) or disabled (0).

`bp -query testadd.vhd 2 59`
Lists all executable lines in *testadd.vhd* between lines 2 and 59.

`bp -c and_gate_init`
Sets a C breakpoint at the entry to C function **and_gate_init**.

`bp -c and_gate.c:46`
Sets a C breakpoint at line 46 in the file *and_gate.c*.

`bp -c 44`
Sets a C breakpoint at line 44 in the current C or SystemC file.

`bp -c *0xff130504`
Sets a C breakpoint at hexadecimal address 0xff130504.

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

add button (CR-52), **bd** (CR-76), **disablebp** (CR-153), **enablebp** (CR-163), **onbreak** (CR-210), **when** (CR-375) , *Chapter 7 - SystemC simulation*, *Chapter 14 - C Debug*

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>`
The directory to which to change. Optional. If no directory is specified, ModelSim changes to your home directory.

cdbg

The **cdbg** command provides command-line equivalents of the menu options that are available for "C Debug" (UM-473). For some of the commands there is a required argument "on | off". The value can be either 'on' or 'off'. For example:

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

Syntax

```
cdbg
  auto_find_bp | debug_on | enable_auto_step <on | off> | init_mode_complete
  | init_mode_setup | interrupt | keep_user_init_bps <on | off> | quit |
  refresh_source_window | set_debugger <path> |
  show_source_balloon <on | off> | stop_on_quit <on | off>
```

Arguments

`auto_find_bp`
Sets breakpoints on all currently known function entry points. See "Finding function entry points with Auto find bp" (UM-479). Equivalent to selecting **Tools** > **C Debug** > **Auto find bp**.

`debug_on`
Enables the C Debugger. Equivalent to selecting **Tools** > **C Debug** > **Start C Debug**.

`enable_auto_step <on | off>`
Enables/disables auto-step mode. See "Identifying all registered function calls" (UM-480). Equivalent to selecting **Tools** > **C Debug** > **Enable auto step**.

`init_mode_complete`
Continues loading the design without stopping at functions calls. See "Debugging functions during elaboration" (UM-483). Equivalent to selecting **Tools** > **C Debug** > **Complete load**.

`init_mode_setup`
Enables initialization mode. See "Debugging functions during elaboration" (UM-483). Equivalent to selecting **Tools** > **C Debug** > **Init mode**.

`interrupt`
Reactivates the C debugger when stopped in HDL code. Equivalent to selecting **Tools** > **C Debug** > **C Interrupt** or clicking the 'C Interrupt' toolbar button.

`keep_user_init_bps <on | off>`
Specifies whether breakpoints set during initialization mode are retained after the design finishes loading. See "Debugging functions during elaboration" (UM-483). Equivalent to toggling the 'Keep user init bps' button in the C Debug setup dialog.

`quit`
Quits the C Debugger. Equivalent to selecting **Tools** > **C Debug** > **Quit C Debug**.

`refresh_source_window`
Re-opens a C source file if you close the Source window inadvertently while stopped in the C debugger. Equivalent to selecting **Tools** > **C Debug** > **Refresh**.

`set_debugger <path>`
Sets the path to your **gdb** installation. The argument path is required and is the complete pathname to the **gdb** executable. For example:

```
cdbg set_debugger_path /usr/bin/gdb
```

`show_source_balloon <on | off>`
Enables/disables the source balloon popup. See "C Debug dialog reference" (UM-490). Equivalent to toggling the 'Show source balloon' button on the C Debug setup dialog.

`stop_on_quit <on | off>`
Enables/disables debugging capability when the simulator is exiting. See "Debugging functions when quitting simulation" (UM-487). Equivalent to toggling the 'Stop on quit' button on the C Debug setup dialog.

change

The **change** command modifies the value of a VHDL constant, generic, or variable; Verilog register or variable; or C variable if running C Debug (UM-473).

Syntax

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

Arguments

<variable>
Specifies the name of one of the following types of objects:

VHDL

- Scalar variables, constants, and generics of all types except FILE
- Scalar subelements of composite variables, constants, and generics of all types except FILE
- One-dimensional arrays of enumerated character types (including slices)
- Access types (an access type pointer can be set to "null"; the value that an access type points to can be changed as specified above)

Verilog

- Parameters
- Registers and memories
- Integer, real, realtime, and time variables
- Subelements of register, integer, real, realtime, and time multi-dimensional arrays (all dimensions must be specified)
- Bit-selects and part-selects of the above except for objects whose basic type is real

C

- Scalar C variables of type int, char, double, or float
- Individual fields of a C structure
- SystemC primitive channels are not supported

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

<value>
Defines a value for the variable. Required. The specified value must be appropriate for the type of the variable.

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 count 16#FFFF`
Changes the value of the variable *count* to the hexadecimal value FFFF.

`change {rega[16]} 0`
Changes the value of the element of *rega* that is specified by the index (i.e., 16).

`change {foo[20:22]} 011`
Changes the value of the set of elements of foo that is specified by the slice (i.e., 20:22).

`change x 1.5`
Sets the value of *x* (type double) to 1.5.

`change a1.c1 0`
Sets the value of structure member *a1.c1* (type int) to 0.

`change val_b "abcdefg"`
Sets *val_b* (type char *) to point to the string "abcdefg".

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

See also

force (CR-176)

change_menu_cmd

The **change_menu_cmd** command changes the command to be executed for a specified menu item label, in the specified menu, in the specified window. The menu path and label must already exist for this command to function. Returns nothing.

Syntax

```
change_menu_cmd
  <window_name> <menu_path> <label> <Cmd>
```

Arguments

`<window_name>`
Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.).

`<menu_path>`
Name of an existing Tk menu widget plus any submenu path. Required.

`<label>`
Current label on the menu item. Required.

`<Cmd>`
New Tcl command to be executed when selected. Required.

See also

add_menu (CR-58), **add_menucb** (CR-60), **add_menuitem** (CR-61), **add_separator** (CR-62), **add_submenu** (CR-63)

check contention add

The **check contention add** command enables contention checking for the specified nodes. The allowed nodes are Verilog nets and VHDL signals of types std_logic and std_logic_vector. Any other node types and nodes that don't have multiple drivers are silently ignored by the command.

Syntax

```
check contention add
  [-r] [-in] [-out] [-inout] [-internal] [-ports] <node_name>
```

Arguments

`-r`
Specifies that contention checking is enabled recursively into subregions. Optional. If omitted, contention check enabling is limited to the current region.

`-in`
Enables checking on nodes of mode IN. Optional.

`-out`
Enables checking on nodes of mode OUT. Optional.

`-inout`
Enables checking on nodes of mode INOUT. Optional.

`-internal`
Enables checking on internal (non-port) items. Optional.

`-ports`
Enables checking on nodes of modes IN, OUT, or INOUT. Optional.

`<node_name>`
Enables checking for the named node(s). Required.

Description

Bus contention checking detects bus fights on nodes that have multiple drivers. A bus fight occurs when two or more drivers drive a node with the same strength and that strength is the strongest of all drivers currently driving the node. The following table provides some examples for two drivers driving a std_logic signal:

driver 1	driver 2	fight
Z	Z	no
0	0	yes
1	Z	no
0	1	yes
L	1	no

driver 1	driver 2	fight
L	H	yes

Detection of a bus fight results in an error message specifying the node and its drivers' current driving values. If a node's drivers later change value and the node is still in contention, a message is issued giving the new values of the drivers. A message is also issued when the contention ends. The bus contention checking commands can be used on VHDL and Verilog designs.

See also

check contention config command (CR-92), **check contention off** command (CR-93)

check contention config

The **check contention config** command allows you to write checking messages to a file (messages display on your screen by default). You may also configure the contention time limit.

Syntax

```
check contention config
  [-file <filename>] [-time <limit>]
```

Arguments

`-file <filename>`
Specifies a file to which to write contention messages. Optional. If this option is selected, the messages are not displayed to the screen.

`-time <limit>`
Specifies a time limit that a node may be in contention. Optional. Contention is detected if a node is in contention for as long as or longer than the limit. The default limit is 0.

See also

check contention add command (CR-90), **check contention off** command (CR-93)

check contention off

The **check contention off** command disables contention checking for the specified nodes.

Syntax

```
check contention off
  [-all] [-r] [-in] [-out] [-inout] [-internal] [-ports] <node_name>
```

Arguments

`-all`
Disables contention checking for all nodes that have checking enabled. Optional.

`-r`
Specifies that contention checking is disabled recursively into subregions. Optional. If omitted, contention check disabling is limited to the current region.

`-in`
Disables checking on nodes of mode IN. Optional.

`-out`
Disables checking on nodes of mode OUT. Optional.

`-inout`
Disables checking on nodes of mode INOUT. Optional.

`-internal`
Disables checking on internal (non-port) items. Optional.

`-ports`
Disables checking on nodes of modes IN, OUT, or INOUT. Optional.

`<node_name>`
Disables checking for the named node(s). Required.

See also

check contention add command (CR-90), **check contention config** command (CR-92)

check float add

The **check float add** command enables float checking for the specified nodes. The allowed nodes are Verilog nets and VHDL signals of type std_logic and std_logic_vector (other types are silently ignored).

You can set a time limit (the default is zero) for float checking using the **-time <limit>** argument to the **check float config** command (CR-95). If you choose to modify the limit, you should do so prior to invoking any **check float add** commands.

Syntax

```
check float add
  [-r] [-in] [-out] [-inout] [-internal] [-ports] <node_name>
```

Arguments

`-r`
Specifies that float checking is enabled recursively into subregions. Optional. If omitted, float check enabling is limited to the current region.

`-in`
Enables checking on nodes of mode IN. Optional.

`-out`
Enables checking on nodes of mode OUT. Optional.

`-inout`
Enables checking on nodes of mode INOUT. Optional.

`-internal`
Enables checking on internal (non-port) items. Optional.

`-ports`
Enables checking on nodes of modes IN, OUT, or INOUT. Optional.

`<node_name>`
Enables checking for the named node(s). Required.

Description

Bus float checking detects nodes that are in the high impedance state for a time equal to or exceeding a user-defined limit. This is an error in some technologies. Detection of a float violation results in an error message identifying the node. A message is also issued when the float violation ends. The bus float checking commands can be used on VHDL and Verilog designs.

See also

check float config command (CR-95), **check float off** command (CR-96)

check float config

The **check float config** command allows you to write checking messages to a file (messages display on your screen by default). You may also configure the float time limit.

Syntax

```
check float config
  [-file <filename>] [-time <limit>]
```

Arguments

`-file <filename>`
Specifies a file to which to write float messages. Optional. If this option is selected, the messages are not displayed to the screen.

`-time <limit>`
Specifies a time limit that a node may be floating. Optional. An error is detected if a node is floating for as long as or longer than the limit. The default limit is 0. Note that you should configure the time limit prior to invoking any **check float add** commands.

See also

check float add command (CR-94), **check float off** command (CR-96)

check float off

The **check float off** command disables float checking for the specified nodes.

Syntax

```
check float off
  [-all] [-r] [-in] [-out] [-inout] [-internal] [-ports] <node_name>
```

Arguments

`-all`
Disables float checking for all nodes that have checking enabled. Optional.

`-r`
Specifies that float checking is disabled recursively into subregions. Optional. If omitted, float check disabling is limited to the current region.

`-in`
Disables checking on nodes of mode IN. Optional.

`-out`
Disables checking on nodes of mode OUT. Optional.

`-inout`
Disables checking on nodes of mode INOUT. Optional.

`-internal`
Disables checking on internal (non-port) items. Optional.

`-ports`
Disables checking on nodes of modes IN, OUT, or INOUT. Optional.

`<node_name>`
Disables checking for the named node(s). Required.

See also

check float add command (CR-94), **check float config** command (CR-95)

check stable off

The **check stable off** command disables stability checking. You may later enable it with **check stable on** (CR-98), and meanwhile, the clock cycle numbers and boundaries are still tracked.

Syntax

```
check stable off
```

Arguments

None.

See also

check stable on command (CR-98)

check stable on

The **check stable on** command enables stability checking on the entire design.

Syntax

```
check stable on
  [-file <filename>] [-period <time>] [-strobe <time>]
```

Arguments

`-file <filename>`
Specifies a file to which to write the error messages. If this option is selected, the messages are not displayed to the screen. Optional.

`-period <time>`
Specifies the clock period (which is assumed to begin at the time the **check stable on** command is issued). Optional. This option is required the first time you invoke the **check stable on** command. It is not required if you later enable checking after it was disabled with the **check stable off** command (CR-97).

`-strobe <time>`
Specifies the elapsed time within each clock cycle that the stability check is performed. Optional. The default strobe time is the period time. If the strobe time falls on a period boundary, then the check is actually performed one timestep earlier. Normally the strobe time is specified as less than or equal to the period, but if it is greater than the period, then the check will skip cycles.

Description

Design stability checking detects when circuit activity has not settled within a period you define for synchronous designs. You specify the clock period for the design and the strobe time within the period during which the circuit must be stable. A violation is detected and an error message is issued if there are pending driver events at the strobe time. The message identifies the driver that has a pending event, the node that it drives, and the cycle number. The design stability checking commands can be used on VHDL and Verilog designs.

Examples

`check stable on -period "100 ps" -strobe "199 ps"`
Performs a stability check 99 ps into each even numbered clock cycle (cycle numbers start at 1).

See also

check stable off command (CR-97)

checkpoint

The **checkpoint** command saves the state of your simulation. The **checkpoint** command saves the simulation kernel state, the *vsim.wlf* file, the list of the HDL items shown in the List and Wave windows, the file pointer positions for files opened under VHDL and the Verilog **$fopen** system task, the states of foreign architectures, and VCD output. Changes you made interactively while running vsim are not saved; for example, macros, virtual objects, command-line interface additions like user-defined commands, and states of graphical user interface windows are not saved. Also, toggle statistics (see the **toggle report** command (CR-275)) are not saved.

Once saved, a checkpoint file may be used with the **restore** command (CR-242) during the same simulation to restore the simulation to a previous state. A VSIM session may also be started with a checkpoint file by using the **vsim -restore** command (CR-357).

Compression of the checkpoint file is controlled by the **CheckpointCompressMode** variable in the *modelsim.ini* file.

If a checkpoint occurs while ModelSim is writing a VCD file, the entire VCD file is copied into the checkpoint file. Since VCD files can be very large, it is possible that disk space problems could occur. Consequently, ModelSim issues a warning in this situation.

Syntax

```
checkpoint
  <filename>
```

Arguments

`<filename>`
Specifies the name of the checkpoint file. Required.

See also

restore (CR-242), **restart** (CR-240), **vsim** (CR-357), "The difference between checkpoint/restore and restart" (UM-85)

compare add

The **compare add** command compares signals in a reference design against signals in a test design. You can specify whether to compare two signals, all signals in the region, or just ports or a subset of ports. Constant signals such as parameters and generics are ignored. See *Chapter 13 - Waveform Compare* for a general overview of waveform comparisons.

The table below shows how compares work between specified reference items and test items.

Reference item	Test item	Result
signal	signal	compare the two signals
signal	region	compare a signal with a name matching the reference signal in the specified test region
region	region	compare all matching signals in both regions
glob expression	signal	legal only if the glob expression selects only one signal
glob expression	region	compare all signals matching the glob expression that match signals in the test region

The **compare add** command supports arguments that specify how each signal state matches std_logic or Verilog values (e.g., -vhdlmatches, see below). Since state matching can also be set on a global basis with the compare options command or PrefCompare() Tcl variables, ModelSim follows state match settings in this order:

1. Use local matching values specified when the compare was created using **compare add** or subsequently configured using **compare configure**.
2. If no local values were set, use global matching values set with the **compare options** command.
3. If no compare options were set, use default matching values specified by PrefCompare Tcl variables.

Syntax

```
compare add
  -clock <name> [-help] [-label <label>] [-list] [-<mode>] [-nowin]
  [-rebuild] [-recursive] [-separator <string>] [-tol <delay>]
  [-tolLead <delay>] [-tolTrail <delay>] [-verbose]
  [-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}]
  [-vlogmatches {<ref-logic-value>=<test-logic-value>:...}]
  [-wavepane <n>] [-wave] [-when {<expression>}] [-win <wname>]
  <referencePath> [<testPath>]
```

Arguments

`-clock <name>`
Specifies the clock definition to use when sampling the specified regions. Required for a clocked comparison; not used for asynchronous comparisons.

`-help`
Lists the description and syntax for the **compare add** command in the Main window transcript. Optional.

`-label <label>`
Specifies a name for the comparison when it is displayed in the Wave window. Optional.

`-list`
Causes specified comparisons to be displayed in the default List window. Optional.

`-<mode>`
Specifies the mode of signal types that are compared. Optional. The actual values the option may take are -in, -out, -inout, -internal, -ports, and -all. You can use more than one mode option in the same command.

`-nowin`
Specifies that compare signals shouldn't be added to any window. Optional. By default, compare signals are added to the default Wave window. See -wave below.

`-rebuild`
Rebuilds a fragmented bus in the test design region and compares it with the corresponding bus in the reference design region. Optional. If a signal is found having the same name as the reference signal, the -rebuild option is ignored. When rebuilding the test signal, the name of the reference signal is used as the wildcard prefix.

`-recursive`
Specifies that signals should also be selected in all nested subregions, and subregions of those, etc. Optional.

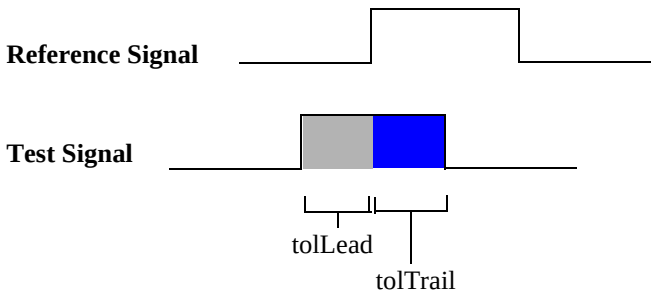
`-separator <string>`
Used with the -rebuild option. When a bus has been broken into bits (bit blasted) by a synthesis tool, ModelSim expects a separator between the base bus name and the bit indication. This option identifies that separator. The default is "_". For example, the signal "mybus" might be broken down into "mybus_0", "mybus_1", etc.

`-tol <delay>`
Specifies the maximum time a test signal edge is allowed to lead or trail a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

`-tolLead <delay>`
Specifies the maximum time a test signal edge is allowed to lead a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

`-tolTrail <delay>`
Specifies the maximum time a test signal edge is allowed to trail a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be placed in curly braces.

Graphical representation of tolLead and tolTrail



`-verbose`
Prints information in the Main window confirming the signals selected for comparison and any type conversions employed. Optional.

`-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}`
Specifies how VHDL signal states in the reference dataset should match values in the test dataset. Optional. Values are specified in a colon-separated list of match values. For example:

```
-vhdlmatches {X=XUD:Z=ZD:1=1HD}
```

Default is {U=UWXD:X=UWXD:0=0LD:1=1HD:Z=ZD:W=UWXD:L=0LD:H=1HD:D=UX01ZWLHD}. The 'D' character represents the '-' "don't care" std_logic value.

`-vlogmatches {<ref-logic-value>=<test-logic-value>:...}`
Specifies how Verilog signal states in the reference dataset should match values in the test dataset. Optional. Values are specified in a colon-separated list of match values. For example:

```
-vlogmatches {0=0:1=1:Z=Z}
```

Default is {0=0:1=1:Z=Z:X=X}.

`-wavepane <n>`
Specifies the pane of the Wave window in which the differences will be viewed. Optional.

`-wave`
Specifies that compare signals be added automatically to the default Wave window. Optional. Default.

`-when {<expression>}`
Specifies a conditional expression that must evaluate to "true" or "1" for differences to be reported. Optional. The expression is evaluated at the start of an observed difference. See "GUI_expression_format" (CR-23) for legal expression syntax.

`-win <wname>`
Specifies a particular window to which to add items. Optional. Used to specify a particular window when multiple instances of that window type exist.

`<referencePath>`
Specifies an absolute or relative path to the reference signal or region, or it can be a glob expression. Required. Relative paths are relative to the current context of the reference dataset. If you specify a glob expression, it will match signals only in the containing context.

`<testPath>`
Specifies an absolute or relative path to the test signal or region. Cannot be a glob expression. Optional. If omitted, the test path defaults to the same path as <referencePath> except for the dataset name.

Examples

`compare add /*`
Selects signals in the reference and test dataset top region according to the default mode. Uses asynchronous comparison with the default tolerances. Assumes that the top regions of the reference and test datasets have the same name and contain the same signals with the same names.

`compare add -port -clock myclock10 gold:.test_ringbuf.ring_inst`
Selects port signals of instance *.test_ringbuf.ring_inst* in both datasets to be compared and sampled on strobe *myclock10*.

`compare add -r gold:/top/cpu test:/testbench/cpu`
Selects all signals in the *cpu* region to be compared asynchronously using the default tolerances. Requires that the reference and test relative hierarchies and signal names within the *cpu* region be identical, but they need not be the same above the *cpu* region.

`compare add -clock clock12 gold:.top.s1`
Specifies that signal *gold:.top.s1* should be sampled at *clock12* and compared with *test:.top.s1*, also sampled at *clock12*.

`compare add -tolLead {3 ns} -tolTrail {5 ns} gold:/asynch/abc/s1 sim:/flat/sigabc`
Specifies that signal *gold:/asynch/abc/s1* should be compared asynchronously with signal *sim:/flat/sigabc* using a leading tolerance of 3 ns and a trailing tolerance of 5 ns.

`compare add -rebuild gold:.counter1.count test:.counter2.cnt`
Causes signals *test:.counter2.cnt_dd* to be rebuilt into bus *test:.counter2.cnt[...]* and compared against *gold:.counter1.count.*

See also

compare annotate (CR-104), **compare clock** (CR-105), **compare configure** (CR-107), **compare continue** (CR-109), **compare delete** (CR-110), **compare end** (CR-111), **compare info** (CR-112), **compare list** (CR-113), **compare options** (CR-114), **compare reload** (CR-118), **compare reset** (CR-119), **compare run** (CR-120), **compare savediffs** (CR-121), **compare saverules** (CR-122), **compare see** (CR-123), **compare start** (CR-125), **compare stop** (CR-127), **compare update** (CR-128), and *Chapter 13 - Waveform Compare*

compare annotate

The **compare annotate** command either flags a comparison difference as "ignore" or adds a text string annotation to the difference. The text string appears when the difference is viewed in error message info popups or in the output of a **compare info** command (CR-112).

Syntax

```
compare annotate
  [-ignore] [-noignore] [-text <message>] <idNum1> [<idNum2>...]
```

Arguments

`-ignore`
Flags the specified difference as "ignore." Optional.

`-noignore`
Undoes a previous -ignore command. Optional.

`-text <message>`
Adds a text string annotation to the difference that is shown wherever the difference is viewed. Optional.

`<idNum1>`
Identifies the difference number to annotate. Required. You can obtain a difference's number using the **compare start** command (CR-125) or a popup dialog. Difference numbers are ordered by time of the difference start, but there may be more than one difference starting at a given time.

`<idNum2>...`
Identifies a second, third, etc. difference number to be annotated in the same way as idNum1. Optional. These are individual references; ranges of numbers cannot be specified.

Examples

`compare annotate -ignore 1 2 10`
Flags difference numbers 1, 2, and 10 as "ignore."

`compare annotate -text "THIS IS A CRITICAL PROBLEM" 12`
Annotates difference number 12 with the message "THIS IS A CRITICAL PROBLEM."

See also

compare add (CR-100), **compare info** (CR-112), and *Chapter 13 - Waveform Compare*

compare clock

The **compare clock** command defines a clock that can then be used for clocked-mode comparisons. In clocked-mode comparisons, signals are sampled and compared only at or just after an edge on some signal.

Syntax

```
compare clock
  [-delete] [-offset <delay>] [-rising | -falling | -both]
  [-when {<expression>}] <clock_name> <signal_path>
```

Arguments

`-delete`
Deletes an existing compare clock. Optional.

`-offset <delay>`
Specifies a time value for delaying the sample time beyond the specified signal edge. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

`-rising`
Specifies that the rising edge of the specified signal should be used. Optional. This is the default.

`-falling`
Specifies that the falling edge of the specified signal should be used. Optional. The default is rising.

`-both`
Specifies that both the rising and the falling edge of the specified signal should be used. Optional. The default is rising.

`-when {<expression>}`
Specifies a conditional expression that must evaluate to "true" or "1" for that clock edge to be used as a strobe. Optional. The expression is evaluated at the time of the clock edge, rather than after the delay has been applied. See "GUI_expression_format" (CR-23) for legal expression syntax.

`<clock_name>`
A name for this clock definition. Required. This name will be used with the compare add command when doing a clocked-mode comparison.

`<signal_path>`
A full path to the signal whose edges are to be used as the strobe trigger. Required.

Examples

`compare clock -rising strobe gold:.top.clock`
Defines a clocked compare strobe named "strobe" that samples signals on the rising edge of signal gold:.top.clock.

`compare clock -rising -delay {12 ns} clock12 gold:/mydesign/clka`
Defines a clocked compare strobe named "clock12" that samples signals 12 ns after the rising edge of signal gold:/mydesign/clka.

See also

compare add (CR-100), *Chapter 13 - Waveform Compare*

compare configure

The **compare configure** command modifies options for compare signals and regions. The modified options are applied to all items in the specified compare path.

Syntax

```
compare configure
  [-clock <name>] [-recursive] [-tol <delay>] [-tolLead <delay>] [-tolTrail
  <delay>] [-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}]
  [-vlogmatches {<ref-logic-value>=<test-logic-value>:...}]
  [-when {<expression>}] <comparePath>
```

Arguments

`-clock <name>`
Changes the strobe signal for the comparison. Optional. If the comparison is currently asynchronous, it will be changed to clocked. This switch may not be used with the -tol, -tolLead, and -tolTrail options.

`-recursive`
Specifies that signals should also be selected in all nested subregions, and subregions of those, etc. Optional.

`-tol <delay>`
Specifies the default maximum time the test signal edge is allowed to trail or lead the reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be in curly braces.

`-tolLead <delay>`
Specifies the maximum time a test signal edge is allowed to lead a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

`-tolTrail <delay>`
Specifies the maximum time a test signal edge is allowed to trail a reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be placed in curly braces.

`-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}`
Specifies how VHDL signal states in the reference dataset should match values in the test dataset. Optional. Values are specified in a colon-separated list of match values. For example:

```
-vhdlmatches {X=XUD:Z=ZD:1=1HD}
```

Default is {U=UWXD:X=UWXD:0=0LD:1=1HD:Z=ZD:W=UWXD:L=0LD:H=1HD:-=UX01ZWLHD}.

`-vlogmatches {<ref-logic-value>=<test-logic-value>:...}`
Specifies how Verilog signal states in the reference dataset should match values in the test dataset. Optional. Values are specified in a colon-separated list of match values. For example:

```
-vlogmatches {0=0:1=1:Z=Z}
```

Default is {0=0:1=1:Z=Z:X=X}.

`-when {<expression>}`
Specifies a conditional expression that must evaluate to "true" or "1" for differences to be reported. Optional. The expression is evaluated at the start of an observed difference. See "GUI_expression_format" (CR-23) for legal expression syntax.

`<comparePath>`
Identifies the path of a compare signal, region, or glob expression. Required.

See also

compare add (CR-100), *Chapter 13 - Waveform Compare*

compare continue

This command is used to continue with comparison difference computations that were suspended using the **compare stop** button or Control-C. If the comparison was not suspended, **compare continue** has no effect.

Syntax

```
compare continue
```

Arguments

None

See also

compare stop (CR-127), *Chapter 13 - Waveform Compare*

compare delete

The **compare delete** command deletes a signal or region from the current open comparison.

Syntax

```
compare delete
  [-recursive] <objectPath>
```

Arguments

`-recursive`
Deletes a region recursively. Optional.

`<objectPath>`
Path in the reference design to the signal or region to be deleted. Required. The dataset prefix is not needed.

See also

compare add (CR-100), *Chapter 13 - Waveform Compare*

compare end

The **compare end** command closes the active comparison without saving any information.

Syntax

```
compare end
```

Arguments

None

See also

compare add (CR-100), *Chapter 13 - Waveform Compare*

compare info

The **compare info** command lists the results of the comparison in the Main window transcript. To save the information to a file, use the -write argument.

Syntax

```
compare info
  [-all] [-count] [-primaryonly] [-signals] [-secondaryonly]
  [<startNum> [<endNum>]] [-summary] [-write <filename>]
```

Arguments

`-all`
Lists all differences (even those marked as "ignore") in the output. Optional. By default, ignored differences are not listed in the output of a compare info command.

`-count`
Returns the total number of primary differences found.

`-primaryonly`
Lists only differences on individual bits, ignoring aggregate values such as a bus. Optional.

`-signals`
Returns a Tcl list of compare signal names that have at least one difference.

`-secondaryonly`
Lists only aggregate value differences such as a bus, ignoring the individual bits.

`<startNum> [<endNum>]`
Specifies the difference numbers to start and end the list with. Optional. If omitted, ModelSim starts the listing with the first difference and ends it with the last. If just **endNum** is omitted, ModelSim ends the listing with the last difference.

`-summary`
Lists only summary information. Optional.

`-write <filename>`
Saves the summary information to <filename> rather than the Main window transcript. Optional.

Examples

`compare info`
Lists all errors in the Main window transcript.

`compare info -summary`
Lists only an error summary in the Main window transcript.

`compare info -write myerrorfile 20 50`
Writes errors 20 through 50 to the file *myerrorfile*.

See also

compare add (CR-100), **compare annotate** (CR-104), *Chapter 13 - Waveform Compare*

compare list

Displays in the Main window a list of all the **compare add** commands currently in effect.

Syntax

```
compare list
  [-expand]
```

Arguments

`-expand`
Expands groups specified by the compare add command to individual signals. Optional.

See also

compare add (CR-100), *Chapter 13 - Waveform Compare*

compare options

The **compare options** command sets defaults for various waveform comparison commands. Those defaults are used when other compare commands are invoked during the current session. To set defaults permanently, edit the appropriate PrefCompare() Tcl variable in the pref.tcl file (see "Preference variables located in Tcl files" (UM-631) for details).

If no arguments are used, compare options returns the current setting for all options. If one option is given that requires a value, and if that value is not given, compare options returns the current value of that option.

Syntax

```
compare options
  [-addwave] [-hide] [-noaddwave] [-show] [-ignoreVlogStrengths]
  [-noignoreVlogStrengths] [-maxsignal <n>] [-maxtotal <n>]
  [-listwin <name>] [-<mode>] [-separator <string>] [-tol <delay>]
  [-tolLead <delay>] [-tolTrail <delay>] [-track] [-notrack]
  [-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}]
  [-vlogmatches {<ref-logic-value>=<test-logic-value>:...}]
  [-wavepane <n>] [-wavewin <name>]
```

Arguments

`-addwave`
Specifies that new comparison objects are added automatically to the Wave window. Optional. Default. You can specify that objects aren't added automatically using the -noaddwave argument. Related Tcl variable is PrefCompare(defaultAddToWave).

`-hide`
Hides all comparisons except those that have at least one difference. Optional. Related Tcl variable is PrefCompare(defaultHideIfNoDiffs).

`-noaddwave`
Specifies that new comparison objects are not added automatically to the Wave window. Optional. The default is to add comparison objects automatically. Related Tcl variable is PrefCompare(defaultAddToWave).

`-show`
Shows all comparisons even if they don't have any differences. Optional. Default. Related Tcl variable is PrefCompare(defaultHideIfNoDiffs).

`-ignoreVlogStrengths`
Specifies that Verilog net strengths should be ignored when comparing two Verilog nets. Optional. Default. Related Tcl variable is PrefCompare(defaultIgnoreVerilogStrengths).

`-noignoreVlogStrengths`
Specifies that Verilog net strengths should *not* be ignored when comparing two Verilog nets. Optional. Related Tcl variable is PrefCompare(defaultIgnoreVerilogStrengths).

`-listwin <name>`
Causes specified comparisons to be displayed in the specified List window. Optional. Related Tcl variable is PrefCompare(defaultListWindow).

`-maxsignal <n>`
Specifies an upper limit for the total differences encountered on any one signal. When that limit is reached, ModelSim stops computing differences on that signal. Optional. The default is 100. Related Tcl variable is PrefCompare(defaultMaxSignalErrors).

`-maxtotal <n>`
Specifies an upper limit for the total differences encountered. When that limit is reached, ModelSim stops computing differences. Optional. The default is 1000. Related Tcl variable is PrefCompare(defaultMaxTotalErrors).

`-<mode>`
Specifies the default mode of signal types that are compared with the **compare add** command (CR-100). Optional. The actual values the option may take are -in, -out, -inout, -internal, -ports, and -all. More than one mode option may be used in the same **compare options** command.

`-separator <string>`
Used with the -rebuild option of the **compare add** command (CR-100). When a bus has been broken into bits (bit blasted) by a synthesis tool, ModelSim expects a separator between the base bus name and the bit indication. This option identifies that separator. The default is "_". For example, the signal "mybus" might be broken down into "mybus_0", "mybus_1", etc. Optional. Related Tcl variable is PrefCompare(defaultRebuildSeparator).

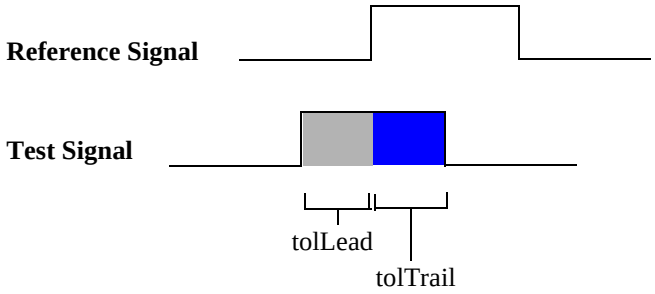
`-tol <delay>`
Specifies the default maximum time the test signal edge is allowed to trail or lead the reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be in curly braces.

You can specify different values for the leading and trailing tolerances using **-tolLead** and **-tolTrail**.

`-tolLead <delay>`
Specifies the default maximum time the test signal edge is allowed to lead the reference edge in an asynchronous comparison. Optional. The default is 0. If a unit (e.g., ps) is used with the time value, the time must be in curly braces. Related Tcl variables are PrefCompare(defaultLeadTolerance) and PrefCompare(defaultLeadUnits).

`-tolTrail <delay>`
Specifies the default maximum time the test signal edge is allowed to trail the reference edge in an asynchronous comparison. Optional. The default is 0. If a unit is used (e.g., ps) with the time value, the time must be in curly braces. Related Tcl variables are PrefCompare(defaultTrailTolerance) and PrefCompare(defaultTrailUnits).

Graphical representation of tolLead and tolTrail



`-track`
Specifies that the waveform comparison should track the current simulation. Optional. Default. The differences will be updated at the end of each **run** command, so if you want to see differences soon after they occur, use many relatively short run commands. Related Tcl variable is PrefCompare(defaultTrackLiveSim).

`-notrack`
Specifies that the waveform comparison should *not* track the current simulation. Optional. Related Tcl variable is PrefCompare(defaultTrackLiveSim).

`-vhdlmatches {<ref-logic-value>=<test-logic-value>:...}`
Specifies how VHDL signal states in the reference dataset should match values in the test dataset. Optional. Values are specified in a colon-separated list of match values. For example:

```
-vhdlmatches {X=XUD:Z=ZD:1=1HD}
```

Default is {U=UWX-:X=UWXD:0=0LD:1=1HD:Z=ZD:W=UWXD:L=0LD:H=1HD:-=UX01ZWLHD}. Related Tcl variable is PrefCompare(defaultVHDLMatches).

`-vlogmatches {<ref-logic-value>=<test-logic-value>:...}`
Specifies how Verilog signal states in the reference dataset should match values in the test dataset. Optional. Values are specified in a colon-separated list of match values. For example:

```
-vlogmatches {0=0:1=1:Z=Z}
```

Default is {0=0:1=1:Z=Z:X=X}. Related Tcl variable is PrefCompare(defaultVLOGMatches).

`-wavepane <n>`
Specifies the default pane of the Wave window in which compare differences will be viewed. Optional. Related Tcl variable is PrefCompare(defaultWavePane).

`-wavewin <name>`
Specifies the default name of the Wave window in which compare differences will be viewed. Optional. Related Tcl variable is PrefCompare(defaultWaveWindow).

Examples

`compare options`
Returns the current value of all options.

`compare options -maxtotal 2000`
Sets the maxtotal option to 2000 differences.

`compare options -maxtotal`
Returns the current value of the maxtotal option.

`compare options -ignoreVlogStrengths`
Sets the option to ignore Verilog net strengths.

`compare options -vlogxmatches {0=0:1=1:Z=Z:X=XZ0}`
Verilog X will now match X, Z, or 0.

`compare options -vhdlmatches {X=UXWD}`
VHDL std_logic X will now match 'U', 'X', 'W', or 'D'.

`compare options -tolLead {300 ps}`
Sets the leading tolerance for asynchronous comparisons to 300 picoseconds.

`compare options -tolTrail {250 ps}`
Sets the trailing tolerance for asynchronous comparisons to 250 picoseconds.

See also

compare add (CR-100), **compare clock** (CR-105), *Chapter 13 - Waveform Compare*

compare reload

The **compare reload** command reloads comparison differences to allow their viewing without recomputation. Prior to invoking compare reload, you must open the relevant datasets with the same names that were used during the original comparison.

Syntax

```
compare reload
  <rulesFilename> <diffsFilename>
```

Arguments

`<rulesFilename>`
Specifies the name of the file that was previously saved using the **compare saverules** command. Required. Must be the first argument.

`<diffsFilename>`
Specifies the name of the file that was previously saved using the **compare savediffs** command. Required.

See also

compare add (CR-100), **compare savediffs** (CR-121), **compare saverules** (CR-122), **compare run** (CR-120), **compare start** (CR-125), *Chapter 13 - Waveform Compare*

compare reset

Clears the current compare differences, allowing another **compare run** command to be executed. Does not modify any of the compare options or any of the signals selected for comparison. This allows you to re-run the comparison with different options or with a modified signal list.

Syntax

```
compare reset
```

Arguments

None

See also

compare add (CR-100), **compare run** (CR-120), and *Chapter 13 - Waveform Compare*

compare run

The **compare run** command runs the difference computation on the signals selected via a **compare add** command. Reports in the Main window the total number of errors found.

Syntax

```
compare run
  [<startTime>] [<endTime>]
```

Arguments

`<startTime>`
Specifies when to start computing differences. Optional. Default is zero. If a unit (e.g., ps) is used with the time value, the time must be in curly braces. The default units are determined by the simulation resolution. (Default simulation resolution is nanoseconds. Simulation resolution can be changed with the **-t** argument of the **vsim** command (CR-357)).

`<endTime>`
Specifies when to end computing differences. Optional. Default is the end of the dataset simulation run that ends earliest. If a unit (e.g., ps) is used with the time value, the time must be placed in curly braces.

Examples

`compare run`
Computes differences over the entire time range.

`compare run {5.3 ns} {57 ms}`
Computes differences from 5.3 nanoseconds to 57 milliseconds.

See also

compare add (CR-100), **compare end** (CR-111), **compare start** (CR-125), *Chapter 13 - Waveform Compare*

compare savediffs

The **compare savediffs** command saves the comparison results to a file that can be reloaded later. To be able to reload the file later, you must also save the comparison setup using the **compare saverules** command.

Syntax

```
compare savediffs
  <diffsFilename>
```

Arguments

`<diffsFilename>`
Specifies the name of the file to create. Required. To load the file at a later time, use the **compare reload** command (CR-118).

See also

compare add (CR-100), **compare reload** (CR-118), **compare saverules** (CR-122), *Chapter 13 - Waveform Compare*

compare saverules

The **compare saverules** command saves the comparison setup information (or "rules") to a file that can be re-executed later. The command saves compare options, clock definitions, and region and signal selections.

Syntax

```
compare saverules
  [-expand] <rulesFilename>
```

Arguments

`-expand`
Expands groups specified by the **compare add** (CR-100) command to individual signals. Optional. If you added a region with the compare add command and then deleted signals from that region, you must use the **-expand** argument or the rules will not reflect the signal deletions.

`<rulesFilename>`
Specifies the name of the file to which you want to save the rules. Required. To load the file at a later time, use the **compare reload** command (CR-118).

See also

compare add (CR-100), **compare reload** (CR-118), **compare savediffs** (CR-121), *Chapter 13 - Waveform Compare*

compare see

The **compare see** command displays the specified comparison difference in the Wave window using whatever horizontal and vertical scrolling are necessary. The signal containing the specified difference will be highlighted, and the active cursor will be positioned at the starting time of the difference.

Syntax

```
compare see
  [-first] [-last] [-next] [-nextanno] [-previous] [-prevanno]
  [-wavepane <n>] [-wavewin <name>]
```

Arguments

`-first`
Shows the first difference, ordered by time. Optional. Performs the same action as the Find First Difference button in the Wave window.

`-last`
Shows the last difference, ordered by time. Optional. Performs the same action as the Find Last Difference button in the Wave window.

`-next`
Shows the next difference (in time) after the currently selected difference. Optional. Performs the same action as the Find Next Difference button in the Wave window.

`-nextanno`
Shows the next annotated difference (in time) after the currently selected difference. Optional. Performs the same action as the Next Annotated Difference button in the Wave window.

`-previous`
Shows the previous difference (in time) before the currently selected difference. Optional. Performs the same action as the Previous Difference button in the Wave window.

`-prevanno`
Shows the previous annotated difference (in time) before the currently selected difference. Optional. Performs the same action as the Previous Annotated Difference button in the Wave window.

`-wavepane <n>`
Specifies the pane of the Wave window in which the difference should be shown. Optional.

`-wavewin <name>`
Specifies the name of the Wave window in which the difference should be shown. Optional.

Examples

`compare see -first`
Shows the earliest difference (in time) in the default Wave window.

`compare see -next`
Shows the next difference (in time) in the default Wave window.

See also

compare add (CR-100), **compare run** (CR-120), *Chapter 13 - Waveform Compare*

compare start

The **compare start** command begins a new dataset comparison. The datasets that you'll be comparing must already be open.

Syntax

```
compare start
  [-batch] [-hide] [-show] [-maxsignal <n>] [-maxtotal <n>]
  [-refDelay <delay>] [-testDelay <delay>] <reference_dataset>
  [<test_dataset>]
```

Arguments

`-batch`
Specifies that comparisons will not be automatically inserted into the Wave window. Optional.

`-hide`
Hides all comparisons except those that have at least one difference. Optional. You can change the default using the **compare options** command (CR-114) or by editing the PrefCompare(defaultHideIfNoDiffs) variable in the *pref.tcl* file.

`-show`
Shows all comparisons even if they don't have any differences. Optional. Default. You can change the default using the **compare options** command (CR-114) or by editing the PrefCompare(defaultHideIfNoDiffs) variable in the *pref.tcl* file.

`-maxsignal <n>`
Specifies an upper limit for the total differences encountered on any one signal. When that limit is reached, ModelSim stops computing differences on that signal. Optional. The default limit is 100. You can change the default using the **compare options** command (CR-114) or by editing the PrefCompare(defaultMaxSignalErrors) variable in the *pref.tcl* file.

`-maxtotal <n>`
Specifies an upper limit for the total differences encountered. When that limit is reached, ModelSim stops computing differences. Optional. The default limit is 1000. You can change the default using the **compare options** command (CR-114) or by editing the PrefCompare(defaultMaxTotalErrors) variable in the *pref.tcl* file.

`-refDelay <delay>`
Delays the reference dataset relative to the test dataset. Optional. If <delay> contains a unit, it must be enclosed in curly braces. Delays are applied to signals specified with the **compare add** command (CR-100). For each signal compared, a delayed virtual signal is created with "_d" appended to the signal name, and these are the signals viewed in the Wave window comparison objects. The delay is not applied to signals specified in compare "when" expressions.

`-testDelay <delay>`
Delays the test dataset relative to the reference dataset. Optional. If <delay> contains a unit, it must be enclosed in curly braces. Delays are applied to signals specified with the **compare add** command (CR-100). For each signal compared, a delayed virtual signal is created with "_d" appended to the signal name, and these are the signals viewed in the Wave window comparison objects. The delay is not applied to signals specified in compare "when" expressions.

`<reference_dataset>`
The dataset to be used as the comparison reference. Required.

`<test_dataset>`
The dataset to be tested against the reference. Optional. If not specified, ModelSim uses the current simulation. The reference and test datasets may be the same.

Examples

```
compare start gold
```
Begins a waveform comparison between a dataset named "gold" and the current simulation. Assumes the gold dataset was already opened.

```
dataset open gold_typ.wlf gold
dataset open bad_typ.wlf test
compare start -maxtotal 5000 -maxsignal 1000 gold test
```
This command sequence opens two datasets and starts a comparison between the two using greater than default limits for total differences encountered.

See also

compare add (CR-100), **compare options** (CR-114), **compare stop** (CR-127), *Chapter 13 - Waveform Compare*

compare stop

This command is used internally by the **compare stop** button to suspend comparison computations in progress. If a **compare run** execution has returned to the VSIM prompt, **compare stop** has no effect. Under Unix, entering a Control-C character in the window that invoked ModelSim has the same effect as **compare stop**.

Syntax

```
compare stop
```

Arguments

None

See also

compare run (CR-120), **compare start** (CR-125), *Chapter 13 - Waveform Compare*

compare update

This command is primarily used internally to update the comparison differences when comparing a live simulation against a .wlf file. The **compare update** command is called automatically at the completion of each simulation run if the "-track" compare option is in effect.

The user can also call **compare update** periodically during a long simulation run to cause difference computations to catch up with the simulation. This command does nothing if the -track compare option was not in effect when the **compare run** command (CR-120) was executed.

Syntax

```
compare update
```

Arguments

None

See also

compare run (CR-120), *Chapter 13 - Waveform Compare*

configure

The **configure** (**config**) 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 (CR-320).

Syntax

```
configure
  list|wave [-window <wname>] [<option> <value>]

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

  [-childrowmargin [<pixels>]] [-cursorlockcolor [<color>]]
  [-gridcolor [<color>]] [-griddelta [<pixels>]] [-gridoffset [<time>]]
  [-gridperiod [<time>]] [-namecolwidth [<width>]] [-rowmargin [<pixels>]]
  [-signalnamewidth [<value>]] [-timecolor [<color>]]
  [-timeline [<value>]] [-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.

`-window <wname>`
Specifies the name of the List or Wave window to target for the **configure** command. (The **view** command (CR-320) 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 (CR-320).

`<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 items 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. See the "GUI_expression_format" (CR-23) for information on expression syntax.

`-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.) See "Setting List window display properties" (UM-293) 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).

`-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).

`-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 item is selected. 1 enables highlighting; 0 disables highlighting. Default is 0. Related Tcl variable is PrefWave(waveSelectEnabled).

To get a more readable listing of all attributes and current values, use the **lecho** (CR-184) command, which pretty-prints a Tcl list.

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

Examples

`config list -strobeperiod`
Displays the current value of the strobeperiod attribute.

`config list -strobeperiod {50 ns} -strobestart 0 -usestrobe 1`
Sets the period of the list strobe and turns it on.

`config wave -vectorcolor blue`
Sets the wave vector color to blue.

`config wave -signalnamewidth 1`
Sets the display in the current Wave window to show only the leaf path of each signal.

See also

view (CR-320), "Preference variables located in Tcl files" (UM-631)

context

The **context** command provides several operations on a context's name. The option you specify determines the operation.

Syntax

```
context dataset | exists | isInst | isNet | isProc | isVar | join | parent |
split | tail | type
  <name>
```

Arguments

`context dataset <name>`
Return the dataset name from the name.

`context exists <name>`
Returns 1 if the name is valid, 0 otherwise.

`context isInst <name>`
Returns 1 if the name is an instance pathname, 0 otherwise.

`context isNet <name>`
Returns 1 if the name is a Signal or Net pathname, 0 otherwise.

`context isProc <name>`
Returns 1 if the name is a Process pathname, 0 otherwise.

`context join <name> <name> ...`
Takes one or more names and combines them, using the correct path separator.

`context parent <name>`
Returns the parent path of the name by removing the tail (see context tail).

`context path <name>`
Returns the pathname portion of the name, removing the dataset name.

`context split <name>`
Returns a list whose elements are the path components in the name. The first element of the list will be the dataset name if one is present in the name, including the dataset separator. For example, `context split /foo/bar/baz` returns `/ foo bar baz .`

`context tail <name>`
Returns all of the characters in the name after the last path separator. If the name contains no separators then returns the name. Any trailing path separator is discarded.

`context type <name>`
Returns a string giving the acc type of the name.

`<name>`
Name of a context object or region. Required. Does not have to be a valid object name unless the specified option requires this (i.e., exists or isInst).

coverage clear

The **coverage clear** command clears all code coverage statement and branch counts obtained during previous run commands and unloads the current exclusion filter file.

Syntax

```
coverage clear
  [<filename>] [-all | -excluded [-user | -pragma | -instance]]
```

Arguments

`<filename>`
Specifies the name of the file you wish to clear. Optional.

`-all`
Clears all statement and branch counts and all user exclusion flags set with the **coverage exclude** command. Optional.

`-excluded`
Unloads a currently loaded exclusion filter file. Exclusion filter files specify files and line numbers that you wish to exclude from Code Coverage statistics. See "Excluding items from coverage" (UM-443) for more details.

`-user`
Clears only user exclusions.

`-pragma`
Clears only pragma exclusions.

`-instance`
Clears only instance-specific exclusions.

Example

`coverage clear -excluded -pragma`
Clears the statement exclusion flags that have been set by the **coverage exclude** command. Only pragma exclusions are cleared.

See also

Chapter 12 - Code Coverage, **coverage exclude** (CR-135), **coverage reload** (CR-136), **coverage report** (CR-137), **coverage save** (CR-140)

coverage exclude

The **coverage exclude** command loads an exclusion filter file. Exclusion filter files specify files and line numbers that you wish to exclude from Code Coverage statistics. (See "Excluding items from coverage" (UM-443) for more details).

Syntax

```
coverage exclude
  <filename>
```

Arguments

`<filename>`
Specifies the file name of the exclusion filter you wish to load. Required. See "Excluding items from coverage" (UM-443) for filter file syntax.

See also

Chapter 12 - Code Coverage, **coverage clear** (CR-134), **coverage reload** (CR-136), **coverage report** (CR-137), **coverage save** (CR-140)

coverage reload

The **coverage reload** command seeds the coverage statistics with the output of a previous **coverage save** command. This allows you to gather statistics from multiple simulation runs.

Syntax

```
coverage reload
  -57<filename> [-incremental] [-install <path>] [-root <new_root_name>]
  [-strip <n>]
```

Arguments

`-57`
Specifies that the file being reloaded was produced in ModelSim version 5.7x. Optional. The coverage file format changed in version 5.8, so you must flag files that are from the earlier version.

`<filename>`
Specifies the file(s) containing data to reload. Required. This file should be the output of a previous **coverage save** command.

`-incremental`
Merges loaded coverage data with current coverage data. Optional. Without this argument, loading coverage data overwrites existing data.

`-install <path>`
Adds <path> as additional hierarchy on the front end of instance and signal names in the data file. Optional. This argument allows you to merge coverage results from simulations that have different hierarchies.

`-root <new_root_name>`
Specifies the root name of the design for which you have a saved coverage report. Optional. This argument has been superseded by the **-strip** and **-install** arguments. It is included for backwards compatibility only.

`-strip <n>`
Removes <n> levels of hierarchy from instance and signal names in the data file. Optional. This argument allows you to merge coverage results from simulations that have different hierarchies.

See also

Chapter 12 - Code Coverage, **coverage clear** (CR-134), **coverage exclude** (CR-135), **coverage report** (CR-137), **coverage save** (CR-140)

coverage report

The **coverage report** command produces textual output of coverage statistics. You can choose from a number of report output options using the arguments listed below. To view this data more interactively, right-click in the *Files* tab of the Main window Workspace and select **Coverage** > **Coverage Reports** from the popup context menu.

Syntax

```
coverage report
  [-above <percent> | -below <percent>] [-append]
  [-byinstance] [-excluded [[-pragma] [-user]] | -totals | [-lines] [-zeros]]
  [-file <filename>] [-instance <pathname>] [-noannotate] [-recursive]
  [-select bces[t|x]] [-source <filename>] [-xml]
```

Arguments

`-above <percent>`
Specifies that only items with coverage values above this percentage be included in the output. Optional.

`-below <percent>`
Specifies that only items with coverage values below this percentage be included in the output. Optional.

`-append`
Appends the current coverage statistics to the named output file (-file <filename>). Optional. Can be used with the **-excluded**, **-instance**, **-lines**, **-total**, and **-zeros** arguments to append specific reports to the output file.

`-byinstance`
Writes out a coverage summary for all instances. Optional.

`-excluded`
Writes out the files and lines that are currently being excluded by the user from the coverage analysis. Shows both pragma and user-based exclusions unless **-pragma** or **-user** are specified. Optional. This is the same information that is shown in the "Current Exclusions pane" (UM-431).

`-pragma`
When used with the **-excluded** argument, writes out *only* lines currently being excluded by pragmas. Optional.

`-user`
When used with the **-excluded** argument, writes out files and lines currently being excluded by the **coverage exclude** command. Optional.

`-file <filename>`
Specifies a file name for the report. Optional. Default is to write the report to the Main window. Environment variables may be used in the pathname.

`-instance <pathname>`
Writes out the source file summary coverage data for the selected instance. Optional.

`-lines`
Writes out the source file summary data and after each file it writes out the details for each executable line in the file. Optional.

`-noannotate`
Produces the same report as **-lines** but removes source code from the output report. Optional.

`-recursive`
Reports on the instance specified with -instance and every included instance, recursively. Can also be used with -lines and -totals. Optional.

`-select bces[t|x]`
Specifies which coverage statistics to include in the report. Optional. By default the report includes statistics for all categories you enabled at compile time.

The characters are as follows:

b–Include branch statistics.

c–Include condition statistics.

e–Include expression statistics.

s–Include statement statistics.

t–Include toggle statistics.

x–Include extended toggle statistics.

`-source <filename>`
Writes a summary of statement coverage data for a specific source file. Optional. Environment variables may be used in the pathname.

`-totals`
Writes out a top-level summary of the number of files, statements, branches, hits, and signal toggles for both file-based and instance-based views of the current analysis. Optional. Useful for tracking changes.

`-xml`
Outputs report in XML format. Optional. See "Reporting coverage data" (UM-446) for more information.

`-zeros`
Writes out a file-based summary of lines that have not been executed (zero hits). Optional. For a detailed report that includes line numbers, use:
coverage report -zeros -lines.

Examples

`coverage report -totals -file myreport.txt`
Writes a top-level summary of the number of files, statements, branches, hits, and signal toggles to *myreport.txt*.

`coverage report -lines -noannotate -select bcs`
Writes detailed branch, condition, and statement statistics, without associated source code, to the specified file.

`coverage report -byinstance`
Writes a summary of code coverage for all instances to the Main window transcript.

`coverage report -lines -byinstance -file myreport.txt`
Writes code coverage details of all instances in the design to *myreport.txt*. The **-lines** option reports coverage statistics for each statement and branch. Branch coverage statistics will following statement statistics and will be presented in four columns: line, column, true branch count, false branch count.

`coverage report -lines -instance /top/p`
Writes code coverage details of one specific instance to the Main window transcript.

`coverage report -excluded -file myexclusions.txt`
Writes both pragma and user-based exclusions to *myexclusions.txt*.

`coverage report -lines -below 90 -file myreport.txt`
Writes a summary of coverage by source file for coverage less than or equal to 90%.

`coverage report -zeros byinstance -file myzerocov.txt`
Writes a list of statements with zero coverage to *myzerocov.txt*.

See also

Chapter 12 - Code Coverage, **coverage clear** (CR-134), **coverage exclude** (CR-135), **coverage reload** (CR-136), **coverage save** (CR-140), **vsim** (CR-357) -coverage option

coverage save

The **coverage save** command saves current coverage statistics to a file that can be reloaded later, preserving instance-specific information.

Syntax

```
coverage save
  [-instance <path>] <filename>
```

Arguments

`-instance <path>`
Saves coverage data for only the specified instance and any of its children, recursively. Optional. <path> is a path to the instance.

`<filename>`
Specifies a file name for the report. Required.

See also

Chapter 12 - Code Coverage, **coverage clear** (CR-134), **coverage exclude** (CR-135), **coverage reload** (CR-136), **coverage report** (CR-137), "$coverage_save(<filename>, [<instancepath>], [<xml_output>])" (UM-149) Verilog system task

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 list (CR-145), **dataset open** (CR-146), **dataset save** (CR-148)

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

```
add wave *
run 100000ns
dataset clear
run 100000ns
```

Clears data in the WLF file from time 0ns to 100000ns, then logs data into the WLF file from time 100000ns to 200000ns.

See also

"WLF files (datasets)" (UM-240), **log** (CR-187)

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 open (CR-146)

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. Required.

See also

dataset alias (CR-141), **dataset list** (CR-145), **dataset open** (CR-146)

dataset list

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

Syntax

```
dataset list
  [-long]
```

Arguments

`-long`
Lists the filename corresponding to each dataset's or alias' logical name. Optional.

See also

dataset alias (CR-141), **dataset save** (CR-148)

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

`dataset open last.wlf test`
Opens the dataset file *last.wlf* and assigns it the logical name *test*.

See also

dataset alias (CR-141), **dataset list** (CR-145), **dataset save** (CR-148), **vsim** (CR-357) -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

```
dataset rename test test2
```
Renames the dataset file "test" to "test2".

See also

dataset alias (CR-141), **dataset list** (CR-145), **dataset open** (CR-146)

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

`dataset save sim gold.wlf`
Saves all current log data in the sim dataset to the file "gold.wlf".

See also

dataset snapshot (CR-149)

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 Main window transcript. 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

`dataset snapshot -size 10`
Creates 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 -mode sequential`
Similar to the previous example but in this case the current WLF file is cleared every time it reaches 10 MB.

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

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.

`dataset snapshot -reset -time 10000`
Enables snapshotting with time=10000 and default mode (cumulative) and default filemode (overwrite).

See also

dataset save (CR-148)

delete

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

Syntax

```
delete
  list|wave [-window <wname>] <item_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 (CR-320) 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 (CR-320).

`<item_name>`
Specifies the name of an item. Required. Must match an item name used in an **add list** (CR-55) or **add wave** (CR-64) command. Multiple item names may be specified. Wildcard characters are allowed.

Examples

```
delete list -window list2 vec2
```
Removes the item *vec2* from the list2 window.

See also

add list (CR-55), **add wave** (CR-64), and "Wildcard characters" (CR-17)

describe

The **describe** command displays information about the specified HDL item, C variable, or design region. The description is displayed in the Main window transcript. The following kinds of items can be described:

- Design region
- **VHDL**
  signals, variables, and constants
- **Verilog**
  nets and registers
- **C**
  variables
- **SystemC**
  signals and ports

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

C variables can be described if you are running "C Debug" (UM-473), and the variables are local to the active call frame for the line in the function in the C source file where you are stopped.

Syntax

```
describe
  <name>
```

Arguments

`<name>`
The name of an HDL item, SystemC signal, or C variable for which you want a description. HDL item names can be full hierarchical names or relative names.

Examples

`describe x`
Prints the type of C variable *x*.

`describe *p`
Prints the type of what *p* points to.

`describe clk prw prdy`
Prints the types of the three specified signals.

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. Note that C breakpoint id#s (see "C Debug" (UM-473)) are prefixed with "c.".

See also

bd (CR-76), **bp** (CR-81), **enablebp** command (CR-163), **onbreak** (CR-210), **resume** (CR-243), **when** (CR-375)

disable_menu

The **disable_menu** command disables the specified menu within the specified window. The disabled menu will become grayed-out and nonresponsive. Returns nothing.

Syntax

```
disable_menu
  <window_name> <menu_path>
```

Arguments

`<window_name>`
Tk path of the window containing the menu. Required. The path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

`<menu_path>`
Name of the Tk menu-widget path. Required.

Examples

```
disable_menu "" File
```
Disables the file menu of the Main window.

```
disable_menu .mywindow File
```
Disables the file menu of the mywindow window.

See also

add_menu (CR-58), **enable_menu** (CR-164)

disable_menuitem

The **disable_menuitem** command disables a specified menu item within the specified menu path of the specified window. The menu item will become grayed-out and nonresponsive. Returns nothing.

Syntax

```
disable_menuitem
  <window_name> <menu_path> <label>
```

Arguments

`<window_name>`
Tk path of the window containing the menu. Required.

Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

`<menu_path>`
Name of the Tk menu-widget path. The path may include a submenu as shown in the example below. Required.

`<label>`
Menu item text. Required.

Examples

```
disable_menuitem .mywindow file.save "Save Results As..."
```
This command locates the mywindow window, and disables the Save Results As... menu item in the save submenu of the file menu.

See also

add_menuitem (CR-61), **enable_menuitem** (CR-165)

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 (CR-212), **onbreak** command (CR-210), or the OnErrorDefaultAction Tcl variable has specified the **resume** command (CR-243).

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 (UM-634) simulator state variable in the macro. See "Making macro parameters optional" (UM-608).

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 (CR-259) to see the other parameters.

Examples

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

`do testfile design.vhd 127`
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.

See also

Chapter 21 - Tcl and macros (DO files), "ModelSim modes of operation" (UM-23), "Using a startup file" (UM-629), DOPATH (UM-613)

down

The **down** command searches for signal transitions or values in the specified List window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a signal takes on a particular value, or an expression of multiple signals evaluates to true. See the **up** command (CR-282) for related functionality.

The procedure for using **down** includes three steps: click on the desired signal; click on the desired starting location; issue the **down** command. (The **seetime** command (CR-257) can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

Syntax

```
down
  [-expr {<expression>}] [-falling] [-noglitch] [-rising]
  [-value <sig_value>] [-window <wname>] [<n>]
```

Arguments

`-expr {<expression>}`
The List window will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced List window. A signal may be specified either by its full path or by the shortcut label displayed in the List window.

See "GUI_expression_format" (CR-23) for the format of the expression. The expression must be placed within curly braces.

`-falling`
Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-noglitch`
Specifies that delta-width glitches are to be ignored. Optional.

`-rising`
Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-value <sig_value>`
Specifies a value of the signal to match. Optional. Must be specified in the same radix that the selected signal is displayed. Case is ignored, but otherwise the value must be an exact string match -- don't-care bits are not yet implemented.

`-window <wname>`
Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the **view** command (CR-320) to change the default window.

`<n>`
Specifies to find the nth match. Optional. If less than n are found, the number found is returned with a warning message, and the marker is positioned at the last match.

Examples

`down -noglitch -value FF23`
Finds the next time at which the selected vector transitions to FF23, ignoring glitches.

`down`
Goes to the next transition on the selected signal.

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the "GUI_expression_format" (CR-23) and can be built with the aid of the "The GUI Expression Builder" (UM-395).

```
down -expr {clk'rising && (mystate == reading) && (/top/u3/addr ==
32'habcd1234)}
```
Searches down for an expression that evaluates to a boolean 1 when signal *clk* just changed from low to high and signal *mystate* is the enumeration reading and signal */top/u3/addr* is equal to the specified 32-bit hex constant.

```
down -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
```
Searches down for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal */top/u3/addr* equals hex ac.

```
down -expr {((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode ==
writing)}
```
Searches down for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, clock just changed from low to high, and signal *mode* is enumeration writing.

See also

"GUI_expression_format" (CR-23), **view** (CR-320), **seetime** (CR-257), **up** (CR-282)

drivers

The **drivers** command displays the names of all drivers of the specified item. The driver list is expressed relative to the top-most design signal/net connected to the specified item. If the item is a record or array, each subelement is displayed individually.

Syntax

```
drivers
  <item_name>
```

Arguments

`<item_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.

See also

readers (CR-236) command

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 Main window.

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

```
echo “The time is    $now ns.”
```
If the current time is 1000 ns, this command produces the message:

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

If the quotes are omitted, all blank spaces of two or more are compressed into one space.

```
echo The time is    $now ns.
```
If the current time is 1000ns, this command produces the message:

```
The time is 1000 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.

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 (CR-207), and the EDITOR (UM-613) 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. Note that C breakpoint id#s (see "C Debug" (UM-473)) are prefixed with "c.".

See also

bd (CR-76), **bp** (CR-81), **disablebp** command (CR-153), **onbreak** (CR-210), **resume** (CR-243), **when** (CR-375), *Chapter 14 - C Debug* (UM-473)

enable_menu

The **enable_menu** command enables a previously-disabled menu. The menu will be changed from grayed-out to normal and will become responsive. Returns nothing.

Syntax

```
enable_menu
  <window_name> <menu_path>
```

Arguments

`<window_name>`
Tk path of the window containing the menu. Required.

Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

`<menu_path>`
Name of the Tk menu-widget path. Required.

Examples

`enable_menu "" File`
Enables the previously-disabled File menu of the Main window.

`enable_menu .mywindow File`
Enables the previously-disabled File menu of the mywindow window.

See also

add_menu (CR-58), **disable_menu** (CR-154)

enable_menuitem

The **enable_menuitem** command enables a previously-disabled menu item. The menu item will be changed from grayed-out to normal, and will become responsive. Returns nothing.

Syntax

```
enable_menuitem
  <window_name> <menu_path> <label>
```

Arguments

`<window_name>`
Tk path of the window containing the menu. Required.

Note that the path for the Main window must be expressed as "". All other window pathnames begin with a period (.) as shown in the example below.

`<menu_path>`
Name of the Tk menu-widget path. The path may include a submenu as shown in the example below. Required.

`<label>`
Menu item text. Required.

Examples

```
enable_menuitem .mywindow file.save "Save Results As..."
```
This command locates the mywindow window and enables the previously-disabled Save Results As... menu item in the save submenu of the file menu.

See also

add_menuitem (CR-61), **disable_menuitem** (CR-155)

environment

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

Syntax

```
environment
  [-dataset] [-nodataset] [[<dataset_prefix>] [<pathname>]]
```

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.

`<dataset_prefix>`
Changes all unlocked windows to the specified dataset context. Optional. The prefix is the logical name of the dataset followed by a colon (e.g., "sim:"). If the `<pathname>` argument is specified as well, it will change the environment to that specified context. If `<pathname>` is omitted, the environment reflects the previously set context. If you don't specify a dataset prefix, then the current dataset is used.

`<pathname>`
Specifies the pathname to which the current region/signal environment is to be changed. Optional. If omitted the command causes the pathname of the current region/signal environment to be displayed.

Multiple levels of a pathname must be separated by the character specified in the PathSeparator (UM-624). A single path separator character can be entered to indicate the top level. Two dots (..) can be entered to move up one level.

Examples

`env`
Displays the pathname of the current region/signal environment.

`env -dataset test`
Changes all unlocked windows to the context of the "test" dataset.

`env test:/top/foo`
Changes all unlocked windows to the context "test: /top/foo".

`env blk1/u2`
Moves down two levels in the design hierarchy.

`env /`
Moves to the top level of the design hierarchy.

examine

The **examine** command examines one or more HDL or SystemC items, and displays current values (or the values at a specified previous time) in the Main window (UM-262). It optionally can compute the value of an expression of one or more items. If you are using C Debug (UM-473), **examine** can display the value of a C variable as well.

The following items can be examined:

- **VHDL**
  signals, shared variables, process variables, constants, and generics
- **Verilog**
  nets, registers, and variables
- **C**
  variables
- **SystemC**
  signals and ports

When stopped in C code, **examine** (with no arguments) displays the values of the local variables and arguments of the current C function.

To display a previous value, specify the desired time using the **-time** option. To compute an expression, use the **-expr** option. The **-expr** and the **-time** options may be used together.

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

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

- 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 item 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 item name.
- If no items of the specified name can be found in the specified context, then an upward search is done to look for a matching item 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 items 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 "HDL and SystemC item names" (CR-12) for more information on specifying names.

Syntax

```
examine
  [-delta <delta>] [-env <path>] [-in] [-out] [-inout] [-internal] [-ports]
  [-expr <expression>] [-name] [-<radix>] [-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 items 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 items that have been logged via the add list, add wave, or log command.

`-env <path>`
Specifies a path to look for a signal name. Optional.

`-expr <expression>`
Specifies an expression to be evaluated. Optional. The items 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" (CR-23) for the format of the expression. The expression must be placed within curly braces.

`-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.

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

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

The **lecho** command (CR-184) will return the output of an examine command in "pretty-print" format. For example,

```
lecho [examine -name clk prw pstrb]
```

`-<radix>`
Specifies the radix for the items that follow in the command. Valid entries (or any unique abbreviations) are: binary, ascii, unsigned, decimal, octal, hex, symbolic, 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 (CR-235).

You can change the default radix permanently by editing the DefaultRadix (UM-623) variable in the *modelsim.ini* file.

`-time <time>`
Specifies the time value between 0 and $now for which to examine the items. Optional. If an expression is specified it will be evaluated at that time. The items 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. This option can be used only with items that have been logged via the add list, add wave, or log command.

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 or SystemC item. Required (except when the **-expr** option is used). All item 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

`examine /top/bus1`
Returns the value of */top/bus1*.

`examine {rega[16]}`
Returns 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 {foo[20:22]}`
Returns the value of the contiguous subelements of *foo* specified by the slice (i.e., 20:22). Note the curly braces.

`examine {/top/\My extended id\ }`
Note that when specifying an item 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 -time {3450 us} -expr {/top/bus and $bit_mask}`
In this example the **-expr** option specifies a signal path and user-defined Tcl variable. The expression will be evaluated at 3450us.

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

Commands like **find** (CR-172) and **examine** return their results as a Tcl list (just a blank-separated list of strings). You can do things like:

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

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

examine -hex [find *]
```

The Tcl variable array, **$examine ()**, can also be used to return values. For example, `$examine (/clk)`. You can also examine an item in the **Source window** (UM-325) by selecting it with the right mouse button.

```
examine x
```
Prints the value of C variable *x*.

```
examine *p
```
Prints the value *p (de-references *p*).

```
examine ip->in1
```
Prints the structure member *in1* pointed to by *ip*.

See also

"HDL and SystemC item names" (CR-12), "Wildcard characters" (CR-17), "GUI_expression_format" (CR-23), *Chapter 14 - C Debug* (UM-473)

exit

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

If you want to stop the simulation using a **when** command (CR-375), you must use a **stop** command (CR-265) within your when statement. DO NOT use an **exit** command or a **quit** command (CR-234). The **stop** command acts like a breakpoint at the time it is evaluated.

Syntax

```
exit
  [-force]
```

Argument

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

find

The **find** command locates items in the design whose names match the name specification you provide. You must specify the type of item 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 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 item names.

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

- 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 item 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 item name.
- If no items of the specified name can be found in the specified context, then an upward search is done to look for a matching item 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 items 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 "HDL and SystemC item names" (CR-12) for more information on specifying names.

Syntax

```
find nets | signals
  [-in] [-inout] [-internal] <item_name> ... [-nofilter] [-out] [-ports]
  [-recursive]

find instances
  [-recursive] <item_name> ...

find virtuals
  [-kind <kind>] [-unsaved] <item_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) items. Optional.

`<item_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

`-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.

`<item_name> ...`
Specifies the instance 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.

`<item_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 signals -r /*`
Finds all signals in the entire design.

`find nets -in /top/xy*`
Finds all input signals in region /top that begin with the letters "xy".

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

`find signals {s[1]}`
Finds a signal named *s1*. Note that you must enclose the item in curly braces because of the square bracket wildcard characters.

`find signals {s[123]}`
Finds signals *s1*, *s2*, or *s3*.

`find signals s(1)`
Finds 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 {/top/data(3 downto 0)}`
Finds 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/\My extended id\ }`
Note that when specifying an item that contains an extended identifier as the last part of the name, there must be a space after the closing '\' and before the closing '}'.

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

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

Additional search options

To search for HDL items within a specific display window, use the **search** command (CR-253) or select **Edit** > **Find**.

See also

"HDL and SystemC item names" (CR-12), "Wildcard characters" (CR-17)

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.

You can force Virtual signals (UM-248) if the number of bits corresponds to the signal value. You cannot force virtual functions. In VHDL and mixed models, you cannot force an input port that is mapped at a higher level or that has a conversion function on the input.

You cannot force bits or slices of a register; you can force only the entire register. You cannot force VHDL or Verilog variables (reg, integer, time, real (or realtime)); these must be changed. See the **change** command (CR-87).

You cannot force a VHDL alias of a VHDL signal.

You cannot force any items within SystemC modules.

Syntax

```
force
  [-freeze | -drive | -deposit] [-cancel <time>] [-repeat <time>] <item_name>
  <value> [<time>] [, <value> <time> ...]
```

Arguments

`-freeze`
Freezes the item at the specified value until it is forced again or until it is unforced with a **noforce** command (CR-204). Optional.

`-drive`
Attaches a driver to the item and drives the specified value until the item is forced again or until it is unforced with a **noforce** command (CR-204). Optional.

This option is illegal for unresolved signals.

`-deposit`
Sets the item to the specified value. The value remains until there is a subsequent driver transaction, or until the item is forced again, or until it is unforced with a **noforce** command (CR-204). Optional.

If one of the **-freeze**, **-drive**, or **-deposit** options is not used, then **-freeze** is the default for unresolved items and **-drive** is the default for resolved items.

If you prefer **-freeze** as the default for resolved and unresolved VHDL signals, change the default force kind in the DefaultForceKind (UM-623) 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.

`-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.

`<item_name>`
Specifies the name of the HDL item to be forced. Required. A wildcard is permitted only if it matches one item. See "HDL and SystemC item names" (CR-12) for the full syntax of an item name. The item 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 item 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 0`
Forces *input1* to 0 at the current simulator time.

`force bus1 01XZ 100 ns`
Forces *bus1* to 01XZ at 100 nanoseconds after the current simulator time.

`force bus1 16#f @200`
Forces *bus1* to 16#F at the absolute time 200 measured in the resolution units selected at simulation start-up.

`force input1 1 10, 0 20 -r 100`
Forces *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 ns, 0 {20 ns} -r 100ns`
Similar to the previous example, but also specifies the time units. Time unit expressions preceding the "-r" must be placed in curly braces.

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

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

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

```
when {/mydut/siga = 10#1} {
  force -deposit /mydut/siga 10#85
}
```
Forces *siga* to decimal value 85 whenever the value on the signal is 1.

See also

noforce (CR-204), **change** (CR-87)

Note: You can configure defaults for the force command by setting the **DefaultForceKind** variable in the *modelsim.ini* file. See "Force command defaults" (UM-630).

gdb dir

The **gdb dir** command sets the source directory search path for the C debugger. See "Setting up C Debug" (UM-475) for more information.

Syntax

```
gdb dir
  [<src_directory_path_1>[:<src_directory_path_2>[:<...>]]]
```

Argument

`<src_directory_path_1>[:<src_directory_path_2>[:<...>]]`
Specifies one or more directories for C source code. Optional. If no directory is specified, the source directory search path is set to the **gdb** default–*$cdir:$cwd*.

Examples

```
gdb dir /a/b/c:~/foo
```
Sets the source directory search path to */a/b/c:~/foo:$cdir:$cwd*

See also

Chapter 14 - C Debug (UM-473)

getactivecursortime

The **getactivecursortime** command gets the time of the active cursor in the Wave window.

Returns the time value.

Syntax

```
getactivecursortime
  [-window <wname>]
```

Arguments

`-window <wname>`
Specifies an instance of the Wave window that is not the default. Otherwise, the default Wave window is used. Optional. Use the **view** command (CR-320) to change the default window.

Examples

`getactivecursortime`
Returns:

```
980 ns
```

See also

left (CR-185), **right** (CR-244)

getactivemarkertime

The **getactivemarkertime** command gets the time of the active marker in the List window.

Returns the time value. If -delta is specified, returns time and delta.

Syntax

```
getactivemarkertime
  [-window <wname>] [-delta]
```

Arguments

`-window <wname>`
Specifies an instance of the List window that is not the default. Otherwise, the default List window is used. Optional. Use the **view** command (CR-320) to change the default window.

`-delta`
Returns the delta value. Optional. Default is to return only the time.

Examples

```
getactivemarkertime -delta
```
Returns:
```
  980 ns, delta 0
```

See also

down (CR-157), **up** (CR-282)

help

The **help** command displays in the Main window 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.

lecho

The **lecho** command takes one or more Tcl lists as arguments and pretty-prints them to the Main window. Returns nothing.

Syntax

```
lecho
  <args> ...
```

Arguments

`<args> ...`
Any Tcl list created by a command or user procedure.

Examples

`lecho [configure wave]`
Prints the Wave window configuration list to the Main window.

left

The **left** command searches left (previous) for signal transitions or values in the specified Wave window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a waveform takes on a particular value, or an expression of multiple signals evaluates to true. See the **right** command (CR-244) for related functionality.

The procedure for using left entails three steps: click on the desired waveform; click on the desired starting location; issue the **left** command. (The **seetime** command (CR-257) can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

Syntax

```
left
  [-expr {<expression>}] [-falling] [-noglitch] [-rising]
  [-value <sig_value>] [-window <wname>] [<n>]
```

Arguments

`-expr {<expression>}`
The waveform display will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced Wave window. A signal may be specified either by its full path or by the shortcut label displayed in the Wave window.

See "GUI_expression_format" (CR-23) for the format of the expression. The expression must be placed within curly braces.

`-falling`
Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-noglitch`
Looks at signal values only on the last delta of a time step. For use with -value option only. Optional.

`-rising`
Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-value <sig_value>`
Specifies a value of the signal to match. Must be specified in the same radix in which the selected waveform is displayed. Case is ignored, but otherwise the value must be an exact string match — don't-care bits are not yet implemented. Only one signal may be selected, but that signal may be an array. Optional.

`-window <wname>`
Specifies an instance of the Wave window that is not the default. Optional. Otherwise, the default Wave window is used. Use the **view** command (CR-320) to change the default window.

`<n>`
Specifies to find the nth match. If less than n are found, the number found is returned with a warning message, and the cursor is positioned at the last match. Optional. The default is 1.

Examples

```
left -noglitch -value FF23 2
```
Finds the second time to the left at which the selected vector transitions to FF23, ignoring glitches.

```
left
```
Goes to the previous transition on the selected signal.

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the "GUI_expression_format" (CR-23) and can be built with the aid of the "The GUI Expression Builder" (UM-395).

```
left -expr {clk'rising && (mystate == reading) && (/top/u3/addr ==
32'habcd1234)}
```
Searches left for an expression that evaluates to a boolean 1 when signal *clk* just changed from low to high and signal *mystate* is the enumeration reading and signal */top/u3/addr* is equal to the specified 32-bit hex constant; otherwise is 0.

```
left -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
```
Searches left for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal */top/u3/addr* equals hex ac.

```
left -expr {((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode ==
writing)}
```
Searches left for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, clock just changed from low to high, and signal *mode* is enumeration writing.

Note: Wave window mouse and keyboard shortcuts (UM-363) are also available for next and previous edge searches. Tab searches right (next) and shift-tab searches left (previous).

See also

"GUI_expression_format" (CR-23), **right** (CR-244), **seetime** (CR-257), **view** (CR-320)

log

The **log** command creates a wave log format (WLF) file containing simulation data for all HDL items whose names match the provided specifications. Items (VHDL signals and variables, Verilog nets and registers, and SystemC primitive channels and ports) that are displayed using the **add list** (CR-55) and **add wave** (CR-64) commands are automatically recorded in the WLF file. The log is stored in a WLF file (formerly a WAV 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** (CR-357) command.

If no port mode is specified, the WLF file contains data for all items in the selected region whose names match the item name specification.

The WLF file is the source of data for the List and Wave windows. An item 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.

Limitations: Verilog memories and VHDL variables can be logged using the variable's full name only.

Syntax

```
log
  [-depth <level>] [-flush] [-howmany] [-in] [-inout] [-internal]
  [-optcells] [-out] [-ports] [-recursive] <item_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) items whose names match the specification. Optional.

`-optcells`
Makes Verilog optimized cell ports visible when using wildcards. Optional. By default Verilog optimized cell ports are not selected even if they match the specified wildcard pattern.

`-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.

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

Examples

```
log -r /*
```
Logs all items in the design.

```
log -out *
```
Logs all output ports in the current design unit.

See also

add list (CR-55), **add wave** (CR-64), **nolog** (CR-205), *Chapter 9 - WLF files (datasets) and virtuals* (UM-239), and "Wildcard characters" (CR-17)

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 0th 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.

macro_option

This command is available for **UNIX only** (excluding Linux).

The **macro_option** command controls the speed and delay of macro (DO file) playback, plus the level of debugging feedback. If invoked without any options, **macro_option** returns all current settings; returns a specific setting if invoked with an option and no argument; returns the previous setting if invoked with both an option and an argument.

Syntax

```
macro_option
  [speed fast | demo] | [delay <delay_time>] | [debug <level>]
```

Arguments

`speed fast | demo`
Set the macro playback speed to fast or demo. Optional.

`delay <delay_time>`
Set the delay time in milliseconds; delay is the time between events in demo mode. Optional.

`debug <level>`
Set the debug level from 1 to 9; 9 giving the most feedback. Optional.

See also

play (CR-214), **run** (CR-246)

mem display

The **mem display** command displays the memory contents of a selected instance to the screen. As a shorthand, if the given instance path only contains 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>] [-dataradix <radix>]
  [-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** (CR-195).

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

`-dataradix <radix>`
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 for the current simulation using the **radix** (CR-235). You can change the default radix permanently by editing the **DefaultRadix** (UM-623) 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 window. Index can be specified.

Examples

`mem display -startaddress 5 -endaddress 10/top/c/mru_mem`
This command displays the memory contents of instance */top/m/mru_mem*, addresses 5 to 10 to the screen as follows:

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

`mem display -format hex -startaddress 5 -endaddress 10 /top/c/mru_mem`
Displays the memory contents of the same instance to the screen in hex format, as follows:

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

See Also

mem load (CR-195)

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
  [-recursive] [<path>]
```

Arguments

`-recursive`
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 window.

Examples

`mem list -r /`
Recursively lists all memories at the top level of the design. Returns:

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

`mem list /top2/uut -r`
Recursively lists all memories in */top2/uut*. 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. 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> [-skip <Nwords>]]
  [-startaddress <st> -endaddress <end>] [<path>]|
```

Arguments

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

`-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. mti is the "MTI memory data file format" (UM-313).

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.

`-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**.

`-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.

`-endaddress <end>`
Specifies the end 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 Workspace area of the Main window.

Memory address indexes can be specified in the instance name also. If addresses are specified both in the instance name and the file, only the intersection of the two address ranges is populated with memory data.

Examples

```
mem load -infile vals.mem -format bin -filltype value -filldata 1‘b0
/top/m/mem
```

Loads 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. 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
```

```
mem load -infile vals.mem -format hex -st 0 -end 12 -filltype value -filldata
16’Hbeef /top/m/mru_mem
```

Loads 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 -filltype value -filldata "16’hab 16’hcd" /top/mem2 -skip 3
```

Loads memory instance */top/mem2* with two words of memory data using the Verilog Hex format, skipping 3 words after each fill pattern sequence.

See also

mem save (CR-198)

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] [-addressadix <radix>] [-dataradix <radix>]
  [-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** (CR-195).

`-addressadix <radix>`
Specifies the address radix for the default mti formatted files. Optional. The <radix> can be specified as: Dec or Hex. The default is the decimal representation.

`-dataradix <radix>`
Specifies the data radix for the default mti formatted files. Optional. The <radix> can be specified as: Symbolic, Binary, Octal, Decimal, Unsigned, or Hex. You can change the default radix for the current simulation using the **radix** (CR-235). You can change the default radix permanently by editing the **DefaultRadix** (UM-623) 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 <st> 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 the memory instance. Optional. The default is the current context, as shown in the Structure window.

Examples

`mem save -format mti -outfile memfile -start 0 -end 10 /top/m/mem`
Saves the memory contents of the instance */top/m/mem(0:10)* to *memfile*, written in the default mti radix. 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 (CR-192), **mem load** (CR-195)

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>] [-dataradix <radix>] [-all]
  [-replace <word>[ <word>...]] [-startaddress <st>] [-endaddress <end>]
  [<path>] -pattern <word>[ <word>...]
```

Arguments

`-addressradix <radix>`
Specifies the radix for the address being displayed. The <radix> can be specified as decimal or hexadecimal. Default is decimal. Optional.

`-dataradix <radix>`
Specifies the radix for the memory data being displayed. Optional. By default, the radix displayed is the system default.

`-all`
Searches 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 <st>`
Specifies the start address for a range of addresses to search. 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 search. The <st> 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 window.

`-pattern <word>[ <word>...]`
Specifies the value of the pattern for the search. Required. Multiple word patterns are accepted, separated by a single white space. Wildcards are accepted in the pattern.

Examples

```
mem search -pattern 16‘Hbeef -dataradix hex /uut/u0/mem3
```

Searches for and prints to the screen all occurrences of the pattern **16‘Hbeef** in */uut/u0/mem3*. Returns:

```
#7845: beef
#7846: beef
#100223: beef
```

```
mem search -p 16‘Hbeef -d hex -replace 16‘Hcafe -st 7846 -end 150000 /uut/u1/mem3
```

Searches for and prints only the first occurrences of **16‘Hbeef** in the address range 7845:150000, replacing them with **16‘Hcafe** in */uut/u0/mem3*. Returns:

```
#7846: cafe
```

```
mem search -p 16‘Hbeef -r 16‘Habe -addressadix hex -all /uut/u1/mem3
```

Replaces all occurrences of **16‘Hbeef** with **16‘Habe** in */uut/u0/mem3*. Returns:

```
#1ea5: 2750
#1ea6: 2750
#1877f: 2750
```

```
mem search -p "*f"
```

Searches for and prints the first occurrence any pattern ending in f.

```
mem search -p "abe cafe" /uut/u1/mem3
```

Searches for and prints the first occurrence of this multiple word pattern.

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>] [-project <project file>]
```

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 (CR-357) is invoked.

`-project <project file>`
Specifies the *.mpf* file to load for this session. Optional.

See also

vsim (CR-357), **do** (CR-156), and "Using a startup file" (UM-629)

next

The **next** command continues a search after you have invoked the **search** command. See "search" (CR-253) for more information.

Syntax

```
next
  <win_type> [-window <wname>]
```

Arguments

`<win_type>`
Specifies structure, signals, process, variables, wave, list, source, or a unique abbreviation thereof. Required.

`-window <wname>`
Specifies an instance of the window that is not the default. Optional. Otherwise, the default window is used. Use the **view** command (CR-320) to change the default window.

noforce

The **noforce** command removes the effect of any active **force** (CR-176) commands on the selected HDL items. The **noforce** command also causes the item's value to be re-evaluated.

Syntax

```
noforce
  <item_name> ...
```

Arguments

`<item_name>`
Specifies the name of a item. Required. Must match an item name used in a previous **force** command (CR-176). Multiple item names may be specified. Wildcard characters are allowed.

See also

force (CR-176) and "Wildcard characters" (CR-17)

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 (CR-187) 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] [<item_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) items 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.

`<item_name>...`
Specifies the item name which you want to unlog. Optional. Multiple item names may be specified. Wildcard characters are allowed.

Examples

`nolog -r /*`
Unlogs all items in the design.

`nolog -reset`
Turns logging back on for all unlogged signals.

See also

add list (CR-55), **add wave** (CR-64), **log** (CR-187)

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. See "Mouse and keyboard shortcuts" (UM-269) for a list of editing shortcuts.

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 in the ModelSim GUI. To open a window, use the **view** command.

Syntax

```
noview
  [*] <window_name>...
```

Arguments

*
  Wildcards can be used, for example: l* (List window), s* (Signal, Source, and Structure windows), even * alone (all windows). Optional.

<window_name>...
  Specifies the ModelSim window type to close. Multiple window types may be used; at least one type (or wildcard) is required. Available window types are:

  `dataflow, list, memory, process, signals, source, structure, variables, and wave`

Examples

`noview wave1`
  Closes the Wave window named "wave1".

`noview l*`
  Closes all List windows.

`noview s*`
  Closes all Structure, Signals, and Source windows.

See also

view (CR-320)

nowhen

The **nowhen** command deactivates selected **when** (CR-375) commands.

Syntax

```
nowhen
  [<label>]
```

Arguments

`<label>`
Specifies an individual when command. Optional. Wildcards may be used to select more than one when command.

Examples

```
when -label 99 b {echo “b changed”}
…
nowhen 99
```
This **nowhen** command deactivates the **when** (CR-375) command labeled 99.

```
nowhen *
```
This **nowhen** command deactivates all **when** (CR-375) commands.

onbreak

The **onbreak** command is used within a macro. 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. Use an empty string to change the **onbreak** command back to its default behavior (i.e., onbreak ""). In that case, the macro will be interrupted after a breakpoint occurs (after any associated **bp** command (CR-81) string is executed).

onbreak commands can contain macro calls.

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. It is an error to execute any commands within an **onbreak** command string following a **run** (CR-246), **run -continue**, or **step** (CR-264) command. This restriction applies to any macros or Tcl procedures used in the **onbreak** command string. Optional.

Examples

```
onbreak {exa data ; cont}
```
Examine the value of the HDL item data when a breakpoint is encountered. Then continue the **run** command (CR-246).

```
onbreak {resume}
```
Resume execution of the macro file on encountering a breakpoint.

```
set broken 0
onbreak {
  set broken 1
  resume
}
run -all
if { $broken } {
  puts "failure"
} else {
  puts "success"
}
```
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 (see "[vsim] simulator control variables" (UM-621)). By default a severity level of failure or above causes a breakpoint; a severity level of error or below does not.

See also

abort (CR-51), **bd** (CR-76), **bp** (CR-81), **do** (CR-156), **onerror** (CR-212), **resume** (CR-243), **status** (CR-263)

onElabError

The **onElabError** command specifies one or more commands to be executed when an error is encountered during elaboration. The command is used by placing it within the *modelsim.tcl* file or a macro. During initial design load **onElabError** may be invoked from within the *modelsim.tcl* file; during a simulation restart **onElabError** may be invoked from 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 (CR-156)

onerror

The **onerror** command is used within a macro; 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 (CR-243) to allow an error message to be printed without halting the execution of the macro file.

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

```
onerror {quit -f}
```
Forces the simulator to quit if an error is encountered while the macro is running.

See also

abort (CR-51), **do** (CR-156), **onbreak** (CR-210), **resume** (CR-243), **status** (CR-263)

Note: You can also set the global **OnErrorDefaultAction** Tcl variable in the *pref.tcl* file to dictate what action ModelSim takes when an error occurs. The onerror command is invoked only when an error occurs in the macro file that contains the onerror command. 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.

pause

The **pause** command placed within a macro interrupts the execution of that macro.

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 (CR-263). 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 (CR-243). To abort the execution of a macro use the **abort** command (CR-51).

See also

abort (CR-51), **do** (CR-156), **resume** (CR-243), **run** (CR-246)

play

This command is available for **UNIX only (excluding Linux)**.

The **play** command replays a sequence of keyboard and mouse actions that were previously saved to a file with the **record** command (CR-237). Returns nothing.

Play returns immediately; the playback proceeds in the background. Caution must be used when putting play commands in do (macro) files.

Syntax

```
play
  <filename>
```

Arguments

`<filename>`
Specifies the recorded file to replay. Required.

Playback controls

The following Tcl **set** commands control the playback type and speed by setting the **play_macro()** global variables. The commands are invoked from the ModelSim command line.

`set play_macro(speed)`
Specify the playback speed: either demo (with the delay specified below), or fast (no delays).

`set play_macro(delay)`
Specifies the delay time in milliseconds. Controls the speed of playback in demo mode.

See also

macro_option (CR-191), **record** (CR-237)

pop

This command is used with C Debug. See *Chapter 14 - C Debug* (UM-473) for more information.

The **pop** command moves the specified number of call frames up the C callstack.

Syntax

```
pop
  <#_of_levels>
```

Arguments

`<#_of_levels>`
Specifies the number of call frames to move up the C callstack. Optional. If unspecified, 1 level is assumed.

Examples

`pop`
Moves up 1 call frame.

`pop 4`
Moves up 4 call frames.

See also

push (CR-231), *Chapter 14 - C Debug* (UM-473)

power add

The **power add** command specifies the signals or nets to track for power information. Data produced by these commands can be translated (by a Synopsys utility) to drive the Synopsys power analysis tools.

The **power add** command is intended to be used as follows:

1. Add the items of interest with the **power add** command.
2. Run the simulation with the **run** command (CR-246).
3. Produce a report with the **power report** command (CR-217).

Syntax

```
power add
  [-in] [-inout] [-internal] [-out] [-ports] [-r] <signalsOrNets> ...
```

Arguments

`-in`
Specifies only inputs. Optional.

`-inout`
Specifies only inouts. Optional.

`-internal`
Specifies only design internal signals or nets. Optional.

`-out`
Specifies only outputs. Optional.

`-ports`
Specifies only design ports. Optional.

`-r`
Searches recursively on a wildcard specified for the signal or net. Optional.

`<signalsOrNets> ...`
Specifies the signal or net to track. Required. Multiple names or wildcards may be used. Must refer to VHDL signals of type bit, std_logic, or std_logic_vector, or to Verilog nets.

When using wildcards, the -in, -inout, -internal, -out, and -ports arguments filter the qualifying signals. If you specify more than one of these arguments, the logical OR of the arguments is performed.

See also

power report (CR-217), **power reset** (CR-218)

See the Synopsys Power documentation for more information.

power report

The **power report** command reports power information for the specified signals or nets. The report can be written to a file or to the Main window. Data produced by these commands can be translated (by a Synopsys utility) to drive the Synopsys power analysis tools.

The **power report** command is intended to be used as follows:

1. Add the items of interest with the **power add** command (CR-216).
2. Run the simulation with the **run** command (CR-246).
3. Produce the report with the **power report** command.

Syntax

```
power report
  [-all] [-noheader] [-file <filename>]
```

Arguments

`-all`
Writes information on all items logged. Optional.

`-noheader`
Suppresses the header to aid in post processing. Optional.

`-file <filename>`
Specifies a filename for the power report. Optional. Default is to write the report to the Main window.

Description

The report format for each line is:

```
signal path, toggle count, hazard count, time at a 1, time at a 0, time at an X
```

- toggle count is the number of 0->1 and 1->0 transitions
- hazard count is the number of 0/1->X, and X->0/1 transitions

  Note that if a signal is initialized at X, and later transitions to 0 or 1, it is not counted as a hazard.
- times are the times spent at each of the three respective states

You will also need to know the total simulation time.

See also

power add (CR-216), **power reset** (CR-218)

See the Synopsys Power documentation for more information.

power reset

The **power reset** command selectively resets power information to zero for the signals or nets specified with the **power add** command (CR-216). Returns nothing.

Syntax

```
power reset
  [-all] [-in] [-inout] [-out] [-internal] [-ports] [-r]
  <signalsOrNets> ...
```

Arguments

`-all`
Resets all signals/nets. Optional.

`-in`
Resets only inputs. Optional.

`-inout`
Resets only inouts. Optional.

`-out`
Resets only outputs. Optional.

`-internal`
Resets only design internal signals or nets. Optional.

`-ports`
Resets only design ports. Optional.

`-r`
Searches recursively on a wildcard specified for the signal or net. Optional.

`<signalsOrNets> ...`
Specifies the signal or net to reset. Required. Multiple names or wildcards may be used.

See also

power add (CR-216), **power report** (CR-217)

See the Synopsys Power documentation for more information.

precision

The **precision** command determines how real numbers display in the graphic interface (e.g., Signals, Wave, Variables, 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

`precision 4`
Results in 4 digits of precision. For example:

```
1.234 or 6543
```

`precision 8#`
Results in 8 digits of precision including trailing zeros. For example:

```
1.2345600 or 6543.2100
```

`precision 8`
Results in 8 digits of precision but doesn't print trailing zeros. For example:

```
1.23456 or 6543.21
```

printenv

The **printenv** command echoes to the Main window 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. Returns nothing. All results go to the Main window.

Syntax

```
printenv
  [<var>...]
```

Arguments

`<var>...`
Specifies the name(s) of the environment variable(s) to print. Optional.

Examples

`printenv`
Prints all environment variable names and their current values. For example,

```
# CC = gcc
# DISPLAY = srl:0.0
...
```

`printenv USER HOME`
Prints the specified environment variables:

```
# USER = vince
# HOME = /scratch/srl/vince
```

profile clear

The **profile clear** command clears any performance data that has been gathered during previous **run** commands. After this command is executed, all profiling data will be reset.

This command has no effect on the current profiling session. The last **profile on** or **profile off** command will still be in effect.

Syntax

```
profile clear
```

Arguments

None

See also

Chapter 11 - Performance Analyzer (UM-407), **profile interval** (CR-222), **profile off** (CR-223), **profile on** (CR-224), **profile option** (CR-225), **profile report** (CR-226)

Note: Profiling must be active when this command is invoked. Use the **profile on** command (CR-224) to begin profiling.

profile interval

The **profile interval** command selects the frequency with which the profiler collects samples during a run command. To use this command, first enable profiling with the **profile on** command (CR-224).

Syntax

```
profile interval
  [<sample_frequency>]
```

Arguments

`<sample_frequency>`
An integer value from 1 to 999 that represents how many milliseconds to wait between each sample collected during a profiled simulation run. Default is 10 ms.

If the sample-frequency is not supplied, the **profile interval** command returns the current sample frequency.

See also

Chapter 11 - Performance Analyzer (UM-407), **profile clear** (CR-221), **profile off** (CR-223), **profile on** (CR-224), **profile option** (CR-225), **profile report** (CR-226)

profile off

The **profile off** command disables runtime profiling.

Syntax

```
profile off
```

Arguments

None

See also

Chapter 11 - Performance Analyzer (UM-407), **profile clear** (CR-221), **profile interval** (CR-222), **profile on** (CR-224), **profile option** (CR-225), **profile report** (CR-226)

profile on

The **profile on** command enables Performance Analyzer, a tool that performs runtime analysis of where your simulation is spending its time. After this command is executed, every subsequent **run** command will be profiled.

See *Chapter 11 - Performance Analyzer* (UM-407) for further details on profiling.

Syntax

```
profile on
```

Arguments

None

See also

Chapter 11 - Performance Analyzer (UM-407), **profile clear** (CR-221), **profile interval** (CR-222), **profile off** (CR-223), **profile option** (CR-225), **profile report** (CR-226)

Example

```
profile on
run 1000 ns
profile report -hier -file perf.rpt
```

This set of commands enables the profiler, runs the simulation for 1000 nanoseconds, and outputs the profiling data to *perf.rpt*.

profile option

The **profile option** command changes how profiling data are reported. To use this command, first enable profiling with the **profile on** command (CR-224).

Syntax

```
profile option
  collapse_sections | raw_data [on | off | status]
```

Arguments

`collapse_sections`
Groups profiling data by section. A section consists of regions of code such as VHDL processes, functions, or Verilog *always* blocks. By default all profiling data are reported on a per line basis. Required if **raw_data** isn't specified.

`raw_data`
Reports the raw number of samples that occurred in a line or a section. By default all profiling results are reported on a percentage basis. Required if **collapse_sections** isn't specified.

`on | off | status`
Specifies whether to enable, disable, or report the status of the profile options. Optional. If omitted, the profile option command acts as a toggle.

See also

Chapter 11 - Performance Analyzer (UM-407), **profile clear** (CR-221), **profile interval** (CR-222), **profile off** (CR-223), **profile on** (CR-224), **profile report** (CR-226)

profile report

The **profile report** command outputs profiling data that have been gathered up to the point that you execute the command. To use this command, first enable profiling using the **profile on** command (CR-224).

See *Chapter 11 - Performance Analyzer* (UM-407) for further details on profiling.

Syntax

```
profile report
  [-hierarchical | -ranked] [-file <filename>] [-cutoff <percentage>]
```

Arguments

`-hierarchical`
Produces a hierarchical report of profiling data in a call-graph style format. Optional. Default.

`-ranked`
Produces a sorted report of profiling data. The ranked format sorts the modules and code lines by simulation time. Optional.

`-file <filename>`
Specifies a file name for the report. Optional. Default is to write the report to the Main window transcript.

`-cutoff <percentage>`
Filter out entries in the report that had less than <percentage> of time spent in them. Optional. Default is to report all entries (i.e., 0%).

See also

Chapter 11 - Performance Analyzer (UM-407), **profile clear** (CR-221), **profile interval** (CR-222), **profile off** (CR-223), **profile on** (CR-224), **profile option** (CR-225)

Example

```
profile on
run 1000 ns
profile report -hier -file perf.rpt
```

This set of commands enables the profiler, runs the simulation for 1000 nanoseconds, and outputs the profiling data to *perf.rpt*.

project

The **project** commands are used to perform common operations on projects. Use this command outside of a simulation session.

Syntax

```
project
  [addfile <filename>] | [close] | [compileall] | [delete <project>] | [env]
  | [history] | [new <home_dir> <proj_name> [<defaultlibrary>]
  [<use_current>]] | [open <project>] | [removefile <filename>]
```

Arguments

`addfile <filename>`
Adds the specified file to the current open project. Optional.

`close`
Closes the current project. Optional.

`compileall`
Compiles all files in the current project. Optional.

`delete <project>`
Deletes a specified project file. Optional.

`env`
Returns the current project file. Optional.

`history`
Lists a history of manipulated projects. Optional.

`new <home_dir> <proj_name> [<defaultlibrary>] [<use_current>]`
Creates a new project under a specified home directory with a specified name and optionally a default library. Optional. If use_current is set to 1, then ModelSim uses the current *modelsim.ini* file when creating the project rather than the default. You must specify a default library if you want to specify use_current.

`open <project>`
Opens a specified project file, making it the current project. Changes the current working directory to the project's directory. Optional.

`removefile <filename>`
Removes the specified file from the current project. Optional.

Examples

`project open /user/george/design/test3/test3.mpf`
Makes */user/george/design/test3/test3.mpf* the current project and changes the current working directory to */user/george/design/test3*.

`project compileall`
Executes current project library build scripts.

property list

The **property list** command changes one or more properties of the specified signal, net, or register in the List window (UM-286). The properties correspond to those you can set by selecting **View** > **Signal Properties** (List window). At least one argument must be used.

Syntax

```
property list
  [-window <wname>] [-label <label>] [-radix <radix>]
  [-trigger <setting>] [-width <number>] <pattern>
```

Arguments

`-window <wname>`
Specifies a particular List window when multiple instances of the window exist (e.g., list2). 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 (CR-320).

`-label <label>`
Specifies the label to appear at the top of the List window column. Optional.

`-radix <radix>`
Specifies the radix for List window items. Optional.

Valid entries (or any unique abbreviations) are: binary, ascii, unsigned, decimal, octal, hex, symbolic, 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 (CR-235). You can change the default radix permanently by editing the DefaultRadix (UM-623) variable in the *modelsim.ini* file.

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

`-trigger <setting>`
Valid settings are 0 or 1. Setting trigger to 1 will enable the List window to be triggered by changes in the items matching the specified pattern. Optional.

`-width <number>`
Valid numbers are 1 through 256. Specifies the desired column width for the items matching the specified pattern. Optional.

`<pattern>`
Specifies a name or wildcard pattern to match the full pathnames of the signals, nets, or registers for which you are defining the property change. Required.

property wave

The **property wave** command changes one or more properties of the specified signal, net, or register in the Wave window (UM-337). The properties correspond to those you can set by selecting **View > Signal Properties** (Wave window). At least one argument must be used.

Syntax

```
property wave
  [-window <wname>] [-color <color>] [-format <format>] [-height <number>]
  [-offset <number>] [-radix <radix>] [-scale <float>] <pattern>
```

Arguments

`-window <wname>`
Specifies a particular Wave window when multiple instances of the window exist (e.g., wave2). 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 (CR-320).

`-color <color>`
Specifies the color to be used for the waveform. Optional.

`-format <format>`
The waveform <format> can be expressed as:

`analog`
Displays a waveform whose height and position is determined by the **-scale** and **-offset** values (shown below). Optional.

`literal`
Displays the waveform as a box containing the item value (if the value fits the space available). Optional.

`logic`
Displays values as 0, 1, X, or Z. Optional.

`-height <number>`
Specifies the height (in pixels) of the waveform. Optional.

`-offset <number>`
Specifies the waveform position offset in pixels. Valid only when **-format** is specified as analog. Optional.

`-radix <radix>`
Specifies the radix for Wave window items. Optional.

Valid entries (or any unique abbreviations) are: binary, ascii, unsigned, decimal, octal, hex, symbolic, 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 (CR-235). You can change the default radix permanently by editing the DefaultRadix (UM-623) variable in the *modelsim.ini* file.

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

`-scale <float>`
Specifies the waveform scale relative to the unscaled size value of 1. Valid only when **-format** is specified as analog. Optional.

`<pattern>`
Specifies a name or wildcard pattern to match the full path names of the signals, nets, or registers for which you are defining the property change. Required.

push

This command is used with C Debug. See *Chapter 14 - C Debug* (UM-473) for more information.

The **push** command moves the specified number of call frames down the C callstack.

Syntax

```
push
  <#_of_levels>
```

Arguments

`<#_of_levels>`
Specifies the number of call frames to move down the C callstack. Optional. If unspecified, 1 level is assumed.

Examples

`push`
Moves down 1 call frame.

`push 4`
Moves down 4 call frames.

See also

pop (CR-215), *Chapter 14 - C Debug* (UM-473)

pwd

The Tcl **pwd** command displays the current directory path in the Main window.

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 Main window transcript. To disable echoing for all commands use the **transcript** command (CR-278) with the **-quietly** option.

See also

transcript (CR-278)

quit

The **quit** command exits the simulator. If you want to stop the simulation using a **when** command (CR-375), you must use a **stop** command (CR-265) within your when statement. DO NOT use an **exit** command (CR-171) or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated.

Syntax

```
quit
  [-f | -force] [-sim]
```

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*).

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** (CR-176), **examine** (CR-167), **change** (CR-87), etc.) as well as for displayed values in the Signals, Variables, Dataflow, List, and Wave windows. You can change the default radix permanently by editing the DefaultRadix (UM-623) variable in the *modelsim.ini* file.

Syntax

```
radix
  [-symbolic | -binary | -octal | -decimal | -hexadecimal |
  -unsigned | -ascii]
```

Arguments

Entries may be truncated to any length. For example, **-symbolic** could be expressed as **-s** or **-sy**, etc. Optional.

Also, **-signed** may be used as an alias for **-decimal**. The **-unsigned** radix will display as unsigned decimal. The **-ascii** radix will display a Verilog item as a string equivalent using 8 bit character encoding.

If no arguments are used, the command returns the current default radix.

readers

The **readers** command displays the names of all readers of the specified item. The reader list is expressed relative to the top-most design signal/net connected to the specified item.

Syntax

```
readers
  <item_name>
```

Arguments

`<item_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 (CR-159) command

record

This command is available for **UNIX only (excluding Linux)**.

The **record** command starts recording a replayable trace of all keyboard and mouse actions. Record and play operations may also be run from the macro-helper menu item of the macro menu. Returns nothing.

Syntax

```
record
  [<filename>]
```

Arguments

`<filename>`
Specifies the file for the saved recording. If <filename> is not specified, the recording terminates.

See also

macro_option (CR-191), **play** (CR-214)

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

`report simulator control`
Displays all simulator control variables.

```
# UserTimeUnit = ns
# RunLength = 100
# IterationLimit = 5000
# BreakOnAssertion = 3
# DefaultForceKind = default
# IgnoreNote = 0
# IgnoreWarning = 0
# IgnoreError = 0
# IgnoreFailure = 0
# CheckpointCompressMode = 1
# NumericStdNoWarnings = 0
# StdArithNoWarnings = 0
# PathSeparator = /
# DefaultRadix = symbolic
# DelayFileOpen = 0
```

`report simulator state`
Displays all simulator state variables. Only the variables that relate to the design being simulated are displayed:

```
# 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

"Preference variables located in INI files" (UM-617), and "Preference variables located in Tcl files" (UM-631)

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 (CR-357) 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. See "Restart command defaults" (UM-630).

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. See *Chapter 6 - Verilog PLI / VPI* for more information on the Verilog PLI/VPI and the *ModelSim FLI Reference* for more information on the FLI.

Syntax

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

Arguments

`-force`
Specifies that the simulation will be restarted without requiring confirmation in a popup window. Optional.

`-noassertions`
Specifies that the current assertions configuration will **not** be maintained after the simulation is restarted. Optional. The default is for assertion settings to be maintained after the simulation is restarted.

`-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 items 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 items 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 items displayed in the Wave window to remain in the window with the same format.

See also

checkpoint (CR-99), **restore** (CR-242), **vsim** (CR-357), "Checkpointing and restoring simulations" (UM-84), "The difference between checkpoint/restore and restart" (UM-85)

restore

The **restore** command restores the state of a simulation that was saved with a **checkpoint** command (CR-99) during the current invocation of VSIM (called a "warm restore").

The items restored are: simulation kernel state, *vsim.wlf* file, HDL items listed in the List and Wave windows, file pointer positions for files opened under VHDL and under Verilog $fopen, and the saved state of foreign architectures.

If you want to **restore** while running VSIM, use this command. If you want to start up VSIM and restore a previously-saved checkpoint, use the **-restore** switch with the **vsim** command (CR-357) (called a "cold restore").

Note: Checkpoint/restore allows a cold restore, followed by simulation activity, followed by a warm restore back to the original cold-restore checkpoint file. Warm restores to checkpoint files that were not created in the current run are not allowed except for this special case of an original cold restore file.

Syntax

```
restore
  <filename>
```

Arguments

`<filename>`
Specifies the name of the checkpoint file. Required.

See also

checkpoint (CR-99), **vsim** (CR-357), "The difference between checkpoint/restore and restart" (UM-85)

resume

The **resume** command is used to resume execution of a macro file after a **pause** command (CR-213) or a breakpoint. It may be input manually or placed in an **onbreak** (CR-210) command string. (Placing a **resume** command in a **bp** (CR-81) command string does not have this effect.) The **resume** command can also be used in an **onerror** (CR-212) 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 (CR-51), **do** (CR-156), **onbreak** (CR-210), **onerror** (CR-212), **pause** (CR-213)

right

The **right** command searches right (next) for signal transitions or values in the specified Wave window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a waveform takes on a particular value, or an expression of multiple signals evaluates to true. See the **left** command (CR-185) for related functionality.

The procedure for using **right** entails three steps: click on the desired waveform; click on the desired starting location; issue the **right** command. (The **seetime** command (CR-257) can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

Syntax

```
right
  [-expr {<expression>}] [-falling] [-noglitch] [-rising]
  [-value <sig_value>] [-window <wname>] [<n>]
```

Arguments

`-expr {<expression>}`
The waveform display will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced Wave window. A signal may be specified either by its full path or by the shortcut label displayed in the Wave window.

See "GUI_expression_format" (CR-23) for the format of the expression. The expression must be placed within curly braces.

`-falling`
Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-noglitch`
Looks at signal values only on the last delta of a time step. For use with the **-value** option only. Optional.

`-rising`
Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-value <sig_value>`
Species a value of the signal to match. Must be specified in the same radix that the selected waveform is displayed. Case is ignored, but otherwise the value must be an exact string match -- don't-care bits are not yet implemented. Only one signal may be selected, but that signal may be an array. Optional.

`-window <wname>`
Specifies an instance of the Wave window that is not the default. Optional. Otherwise, the default Wave window is used. Use the **view** command (CR-320) to change the default window.

`<n>`
Specifies to find the nth match. If less than n are found, the number found is returned with a warning message, and the cursor is positioned at the last match. Optional. The default is 1.

Examples

`right -noglitch -value FF23 2`
Finds the second time to the right at which the selected vector transitions to FF23, ignoring glitches.

`right`
Goes to the next transition on the selected signal.

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the "GUI_expression_format" (CR-23) and can be built with the aid of the "The GUI Expression Builder" (UM-395).

```
right -expr {clk'rising && (mystate == reading) && (/top/u3/addr ==
32'habcd1234)}
```
Searches right for an expression that evaluates to a boolean 1 when signal *clk* just changed from low to high and signal *mystate* is the enumeration reading and signal */top/u3/addr* is equal to the specified 32-bit hex constant; otherwise is 0.

```
right -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
```
Searches right for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal */top/u3/addr* equals hex ac.

```
right -expr {((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode ==
writing)}
```
Searches right for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, and clock just changed from low to high and signal *mode* is enumeration writing.

Note: Wave window mouse and keyboard shortcuts (UM-363) are also available for next and previous edge searches. Tab searches right (next) and shift-tab searches left (previous).

See also

"GUI_expression_format" (CR-23), **left** (CR-185), **seetime** (CR-257), **view** (CR-320)

run

The **run** command advances the simulation by the specified number of timesteps.

Syntax

```
run
  [<timesteps>[<time_units>]] | [-all] | [-continue] | [-finish] | [-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). See "Setting default simulation options" (UM-386). Time steps and time units may also be set with the RunLength (UM-625) and UserTimeUnit (UM-626) 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** (CR-264) command, **step -over** command or a breakpoint. A **run -continue** command may be input manually or used as the last command in a **bp** (CR-81) command string. Optional.

`-finish`
In "C Debug" (UM-473) only, continues the simulation run and returns control to the calling function. 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

`run 1000`
Advances the simulator 1000 timesteps.

`run 10.4 ms`
Advances the simulator the appropriate number of timesteps corresponding to 10.4 milliseconds.

```
run @8000
```
Advances the simulator to timestep 8000.

See also

step (CR-264)

sccom

The **sccom** command compiles SystemC source code into the **work** library. The **sccom** compiler interacts with a C/C++ compiler to compile and link designs.

This command may be invoked from within ModelSim or from the operating system command prompt. This command may also be invoked during simulation.

Compiled libraries are platform dependent. If you move between platforms, you will need to run **vdel -allsystemc** on the working library and then recompile your SystemC source.

Compiled libraries are version dependent. For example, you cannot use a library compiled with 5.8 in a simulation using 5.8a **vsim**. You have to re-compile your design with the updated version of **sccom**.

Certain restrictions apply when compiling SystemC modules with HP aCC. See "Restrictions on compiling with HP aCC" (UM-194) for details.

During the linking of the design (with **sccom -link**) the order in which you specify archives (*.a*) and object files is very important. Any dependent *.a* or *.o* must be specified before the *.a* or *.o* on which it depends.

Syntax

```
sccom
  [<CPP compiler options>] [-help] [-link] [-log <logfile>] [-nologo]
  [-nonamebind] [-scv] [-vv] [-verbose] [-version] <filename>
```

Arguments

`<CPP compiler options>`
Any normal C++ compiler option can be used, with the exception of the **-o** and **-c** options. By default, **sccom** compiles without debugging information. Specify the **-g** argument to compile for debugging. You can specify arguments for all ModelSim runs by editing the **CppOptions** variable in the *modelsim.ini* file.

`-help`
Displays the command's options and arguments. Optional.

`-link`
Performs the final link of all previously compiled SystemC source code. Required before running simulation. Any dependent *.a* or *.o* must be specified before the *.a* or *.o* on which it depends. Two types of dependencies are possible, and where you place the **-link** argumnent is different based on which type of dependency the files have.

If your archive or object is dependent on the *.o* files created by **sccom** (i.e. your code references symbols in the generated SystemC *.o* files), then you must specify the **-link** argument after the list of files, as shown below:

```
sccom a.o b.o libtemp.a -link
```

Under the covers, the C++ linker's command and argument order looks like this:

```
ld a.o b.o libtemp.a <internal list of SC .o files> libsystemc.a
```

However, if the *.o* files created by **sccom** are dependent on the object or archive you provided, then the **-link** argument must be placed after the object files or archive:

```
sccom -link a.o b.o libtemp.a
```

In this case, the "undercover" command and argument order looks like this:

```
ld <internal list of SC .o files> libsystemc.a a.o b.o libtemp.a
```

`-log <logfile>`
Specifies the logfile in which to collect output. Optional. Related *modelsim.ini* variable is **sccomLogfile**.

`-nologo`
Disables the startup banner. Optional.

`-nonamebind`
Disables the automatic name binding for all modules declared in header files. Optional. If a module declared in a C++ source file uses the SC_CTOR macro, then automatic name binding should be disabled. For more information, see "Name association (binding)" (UM-204). Related *modelsim.ini* variable is **NoNameBind**.

Important: If you use **-nonamebind** for one compilation, you must use it for all subsequent compilations of that design, or a fatal error will occur.

`-scv`
Includes the SystemC verification library. Optional. If you specify this argument when compiling your C code with **sccom**, you must also specify it when linking the object files with **sccom -link**. Related *modelsim.ini* variable is **UseSvc**.

`-vv`
Prints all subprocess invocation information. Optional. An example is the call to **gcc** along with the command-line arguments.

`-verbose`
Prints the name of each sc_module encountered during compilation. Optional. Related *modelsim.ini* variable **sccomVerbose**.

`-version`
Displays the version of **sccom** used to compile the design. Optional.

`<filename>`
Specifies the name of a file containing the SystemC/C++ source to be compiled. Required. Multiple filenames separated by spaces can be entered, or wildcards can be used (e.g., **.cpp*).

Examples

`sccom -g example.cpp`
Compiles *example.cpp* with debugging information.

`sccom -link`
Links the *example.vhd*.

`sccom -I/home/systemc/include -DSC_INCLUDE_FX -g a.cpp b.cpp`
Compiles the SystemC code with an include directory and the compile time macro (SC_INCLUDE_FX) to compile the source with support for fixed point types. For more information, see "Fixed point types" (UM-205).

`sccom -02 a.cpp`
Compiles with the g++ -02 optimization argument.

`sccom -L home/libs/ -l mylib -link`
Links in the library *libmylib.a* when creating the *.so* file. The -L argument specifies the search path for the libraries.

See also

Chapter 7 - SystemC simulation, **vdel -allsystemc** command (CR-315)

scgenmod

Once a Verilog or VHDL module is compiled into a library, you can use the **scgenmod** command to write its equivalent SystemC foreign module declaration to standard output. Optional arguments allow you to generate sc_bit, sc_bv, or resolved port types; sc_logic and sc_lv port types are generated by default.

Syntax

```
scgenmod
  [-help] [-lib <library_name>] [-b] [-r] [-s] <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 **scgenmod** to generate sc_bit or sc_bv port types. Optional.

`-r`
Causes **scgenmod** to generate resolved port types. Optional.

`-s`
Used for the explicit declaration of default sc_logic and sc_lv port types. This is the default. Optional.

`<module_name>`
Specifies the name of the Verilog/VHDL 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 vcounter (clock, topcount, count);
    input clock;
    input topcount;
    output count;

    reg count;
    ...
endmodule
```

After compiling using **vlog** (CR-345), you invoke **scgenmod** on the compiled module with the following command:

```
scgenmod vcounter
```

The SystemC foreign module declaration for the above Verilog module is:

```
class vcounter : public sc_foreign_module
{
    public:
    sc_in<sc_logic> clock;
    sc_in<sc_logic> topcount;
    sc_out<sc_logic> count;

vcounter(sc_module_name nm, const char* hdl_name)
    : sc_foreign_module(nm, hdl_name),
        clock("clock"),
        topcount("topcount"),
        count("count")
    {}
    ~vcounter()
    {}
};
```

See also

Chapter 7 - SystemC simulation

search

The **search** command searches the specified window for one or more items matching the specified pattern(s). The search starts at the item currently selected, if any; otherwise it starts at the window top. The default action is to search downward until the first match, then move the selection to the item found, and return the index of the item found. The search can be continued using the **next** command.

Returns the index of a single match, or a list of matching indices. Returns nothing if no matches are found.

Syntax

```
search
  <win_type> [-window <wname>] [-all] [-field <n>] [-toggle]
  [-forward | -backward] [-wrap | -nowrap] [-exact] [-regexp] [-nocase]
  [-count <n>] <pattern>
```

Arguments for all windows

`<win_type>`
Specifies structure, signals, process, variables, wave, list, source, or a unique abbreviation thereof. Required.

`-window <wname>`
Specifies an instance of the window that is not the default. Optional. Otherwise, the default window is used. Use the **view** command (CR-320) to change the default window.

`-forward`
Search in the forward direction. Optional. This is the default.

`-backward`
Search in the reverse direction. Optional. Default is forward.

`<pattern>`
String or glob-style wildcard pattern. Required. Must be the last argument specified.

Arguments, for all EXCEPT the Source window

`-all`
Finds all matches and returns a list of the indices of all items that match. Optional.

`-field <n>`
Selects different fields to test, depending on the window type:

Window	n=1	n=2	n=3	default
structure	instance	entity/ module	architecture	instance
signals	name	-	cur. value	name
process	status	process label	fullpath	fullpath

Window	n=1	n=2	n=3	default
variables	name	-	cur. value	name
wave	name	-	cur. value	name
list	label	fullname	-	label

Default behavior for the List window is to attempt to match the label and if that fails, try to match the full signal name.

`-toggle`
Adds signals found to the selection. Does not do an initial clear selection. Optional. Otherwise deselects all and selects only one item.

`-wrap`
Specifies that the search continue from the top of the window after reaching the bottom. Optional. This is the default.

`-nowrap`
Specifies that the search stop at the bottom of the window and not continue searching at the top. Optional. The default is to wrap.

Arguments, Source window only

`-exact`
Search for an exact match. Optional.

`-regexp`
Use the pattern as a Tcl regular expression. Optional.

`-nocase`
Ignore case. Optional. Default is to use case.

`-count <n>`
Search for the nth match. Optional. Default is to search for the first match.

Description

With the **-all** option, the entire window is searched, the last item matching the pattern is selected, and a Tcl list of all corresponding indices is returned.

With the **-toggle** option, items found are selected in addition to the current selection.

For the List window, the search is done on the names of the items listed, that is, across the header. To search for values of signals in the List window, use the **down** command (CR-157) and **up** command (CR-282). Likewise, in the Wave window, the search is done on signal names and values in the values column. To search for signal values in the waveform pane of the Wave window, use the **right** command (CR-244) and the **left** command (CR-185). You can also select **Edit > Search** in both windows.

See also

find (CR-172), **next** (CR-203), **view** (CR-320)

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>
```

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" (CR-23) 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 (CR-339), **virtual log** (CR-331), **virtual nolog** (CR-334)

seetime

The **seetime** command scrolls the List or Wave window to make the specified time visible. For the List window, a delta can be optionally specified as well.

Returns nothing

Syntax

```
seetime
  list|wave [-window <wname>] [-select] [-delta <num>] <time>
```

Arguments

`list|wave`
Specifies the target window type. Required.

`-window <wname>`
Specifies an instance of the Wave or List window that is not the default. Optional. Otherwise, the default Wave or List window is used. Use the **view** command (CR-320) to change the default window.

`-select`
Also moves the active cursor or marker to the specified time (and optionally, delta). Optional. Otherwise, the window is only scrolled.

`-delta <num>`
For the List window when deltas are not collapsed, this option specifies a delta. Optional. Otherwise, delta 0 is selected.

`<time>`
Specifies the time to be made visible. Required.

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 (CR-281)

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 (UM-634) 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 (CR-156)

show

The **show** command lists HDL items and subregions visible from the current environment. The items listed include:

- **VHDL**
  signals and instances
- **Verilog**
  nets, registers, tasks, functions, instances, and memories

If using "C Debug" (UM-473), **show** displays the names and types of the local variables and arguments of the current C function.

The **show** command returns formatted results to stdout. To eliminate formatting (to use the output in a Tcl script), use the **Show** command instead.

Syntax

```
show
  [-all] [<pathname>]
```

Arguments

`-all`
Display all names at and below the specified path recursively. Optional.

`<pathname>`
Specifies the pathname of the environment for which you want the items and subregions to be listed. Optional; if omitted, the current environment is assumed.

Examples

`show`
Lists the names of all the items and subregion environments visible in the current environment.

`show /uut`
Lists the names of all the items and subregions visible in the environment named /uut.

`show sub_region`
Lists the names of all the items and subregions visible in the environment named sub_region which is directly visible in the current environment.

See also

environment (CR-166), **find** (CR-172)

simstats

The **simstats** command returns performance-related statistics about the simulation.

If executed without arguments, the command returns a list of pairs like the following:

```
{memory 57376} {{working set} 56152} {time 0} {{cpu time} 0} {context 0} /
{{page faults} 0}
```

See the arguments below for descriptions of each pair.

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 HP-UX, the memory size is reported in the number of pages
- 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 vsim. Optional.

`working`
Returns the portion of allocated virtual memory that is currently being used by all vsim processes. Optional. If this number exceeds the actual memory size, you will encounter performance degradation.

`time`
Returns the cumulative "wall clock time" of the run commands. Optional.

`cpu`
Returns the cumulative processor time of the run commands. Optional. Processor time differs from wall clock time in that processor time is only counted when the cpu is actually running vsim. If vsim is swapped out for another process, cpu time does not increase.

`context`
Returns the number of context swaps (vsim being swapped out for another process) that have occurred during the run commands. Optional.

`faults`
Returns the number of page faults that have occurred during the run commands. Optional.

splitio

The **splitio** command operates on a VHDL inout or out port to create a new signal having the same name as the port suffixed with "__o". The new signal mirrors the output driving contribution of the port.

Note: In ModelSim versions prior to 5.5c, **splitio** was used to split the VHDL inout or output ports so you could re-simulate your design from a vcd file using **vsim -vcdread**. In later versions, addition of the **vcd dumpports** command (CR-287) eliminated the need for **splitio**.

Syntax

```
splitio
  [-outalso | -outonly] [-r] <signal_name>...
```

Arguments

`-outalso`
Allows **splitio** to work on out ports as well as inout ports. Optional.

`-outonly`
Allows **splitio** to work *only* on out ports. Optional.

`-r`
Specifies that the port selection occurs recursively into subregions. Optional. If omitted, included ports are limited to the current region.

`<signal_name>...`
Specifies the VHDL port. Operates only on inout ports by default; out ports may be specified with the options above. Separate multiple port names with spaces. Required. Wildcards can be used.

Examples

The **splitio** command operates on inout or out ports and silently ignores any other signals specified. The new signals created may be specified in any **vsim** (CR-357) commands that operate on signals. These signals appear to be out ports to the signal selection options on **vsim** commands. For example,

`splitio /data`
Creates a signal *data__o* if *data* is an inout port.

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** (CR-210) or **onerror** (CR-212) 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** (CR-243), 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 (CR-51), **do** (CR-156), **pause** (CR-213), **resume** (CR-243)

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 Variables 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 (CR-246)

stop

The **stop** command is used with the **when** command (CR-375) 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.

Use the **run** command (CR-246) with the **-continue** option to continue the simulation run, or the **resume** command (CR-243) 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 (CR-375), you must use a **stop** command within your when statement. DO NOT use an **exit** command (CR-171) or a **quit** command (CR-234). The **stop** command acts like a breakpoint at the time it is evaluated.

See also

bp (CR-81), **resume** (CR-243), **run** (CR-246), **when** (CR-375)

tb

The **tb** (traceback) command displays a stack trace for the current process in the Main window. This lists the sequence of HDL function calls that have been entered to arrive at the current state for the active process.

If you are using "C Debug" (UM-473), **tb** displays a stack trace of the C call stack.

Syntax

```
tb
  [<#_of_levels>]
```

Arguments

`<#_of_levels>`
Specifies the number of call frames in the C stack to display. Optional. If you don't specify a level, the entire C stack is displayed. Argument is available only for "C Debug" (UM-473).

tcheck_set

The **tcheck_set** command works in tandem with **tcheck_status** (CR-269) to report on and enable/disable individual timing checks. **tcheck_set** modifies either a check's reporting or X-generation status and reports the new setting in the Main window transcript.

Disabling a timing check's reporting prevents generation of associated violation messages. For Verilog modules this means ModelSim disables message reporting. For VHDL design units this means ModelSim sets the MsgOn parameter in a VITAL timing check procedure (TCP) to FALSE. Disabling a timing check's X generation removes a timing check's ability to affect the outputs of the simulation. For Verilog modules this means ModelSim toggles the timing check's notifier. For VHDL design units this means ModelSim sets the Xon parameter in a VITAL TCP to FALSE.

tcheck_set *does not* override the effects of invoking **vlog** (CR-345) or **vsim** (CR-357) with the **+nospecify**, **+notimingchecks**, or **+no_neg_tchk** argument. **tcheck_set** can override the effects of invoking **vsim** (CR-357) with the **+no_notifier**, **+no_tchk_msg**, **-g**, or **-G** argument. These latter arguments establish initial values for the simulation, and those values can be modified by **tcheck_set**.

Keep in mind the following if you are using VHDL VITAL:

- VITAL does not provide the granularity to set individual period or width checks. These checks are part of a single VITAL TCP, and **tcheck_set** toggles MsgOn and Xon for all checks in the TCP. See "Examples" below for further information.
- If an instance is not Level-1 optimized, you cannot set values for individual TCPs. You can set values only for the entire instance. **tcheck_status** reports "ALL" for instances that aren't Level-1 optimized. See "Examples" below for further information.

Syntax

```
tcheck_set
  [-quiet] [-r] <instance> [<tcheck>] <Stat>|<MsgStat> <XStat>
```

Arguments

`-quiet`
Suppresses printing the new setting to the Main window transcript. Optional.

`-r`
Attempts to change all checks on this instance and instances below this instance. Optional.

`<instance>`
Specifies the instance for which you want to change the reporting or X-generation status. Required.

`<tcheck>`
Specifies a specific timing check to change. Optional. If you don't specify **<tcheck>** or **-r**, or you specify *ALL* for **<tcheck>**, ModelSim attempts to apply the change to all timing checks in the instance.

You can specify either the integer that is assigned to each timing check (and reported via **tcheck_status**) or the actual timing check name enclosed in double quotes (see "Examples" below). Note that the integer number may change between library compiles.

`<Stat>`
Enables/disables both X generation and violation message reporting for the specified timing check(s). Required unless you specify <**MsgStat**> and <**XStat**>. Specify either ON (enable) or OFF (disable).

`<MsgStat>`
Enables/disables violation message reporting for the specified timing check(s). Required unless you specify <**Stat**>. Specify either ON (enable) or OFF (disable).

`<XStat>`
Enables/disables X generation for the specified timing check(s). Required unless you specify <**Stat**>. Specify either ON (enable) or OFF (disable).

Examples

```
tcheck_set top.y1.u2 "( WIDTH (negedge  CLK) )" OFF
```
Turns off message reporting and X generation for the "(WIDTH (negedge CLK))" check in instance *top.y1.u2*. Creates the following output in the Transcript pane:

```
  #0 ( WIDTH (negedge  CLK) ) MsgOff XOff
```

```
tcheck_set top.y1.u2 1 OFF ON
```
Turns off message reporting for timing check number 1 in instance *top.y1.u2*. Creates the following output in the Transcript pane:

```
  #1 ( WIDTH (posedge  CLK) ) MsgOff XOn
```

```
VSIM 2> tcheck_status dff1
# 1 ( PERIOD CLK ) MsgOn, XOn
#     ( WIDTH (posedge CLK) ) MsgOn, XOn
#     ( WIDTH (negedge CLK) ) MsgOn, XOn

VSIM 3> tcheck_set dff1 "( WIDTH (posedge CLK) )" off on
# 1 ( PERIOD CLK ) MsgOff, XOn
#     ( WIDTH (posedge CLK) ) MsgOff, XOn
#     ( WIDTH (negedge CLK) ) MsgOff, XOn
```
Shows how period and hold checks work with VHDL VITAL. In this case, specifying "off on" for `( WIDTH (posedge CLK) )` also sets `( PERIOD CLK )` and `( WIDTH (negedge CLK) )` to the same values.

```
VSIM 3> tcheck_status dff5
# ALL MsgOn XOn

VSIM 4> tcheck_set dff5 on off
# ALL MsgOn XOff
```
Instance *dff5* is from an unaccelerated model so **tcheck_set** can only toggle message reporting and X generation for all checks on the instance.

See also

tcheck_status (CR-269), "VITAL compliance warnings" (UM-92), *Chapter 17 - Standard Delay Format (SDF) Timing Annotation*, "Disabling timing checks" (UM-555), **-g, -G, no_notifier**, **+no_tchk_msg**, **+nospecify**, **+no_neg_tchk**, and **+notimingchecks** arguments to the **vsim** command (CR-357)

tcheck_status

The **tcheck_status** command works in tandem with **tcheck_set** (CR-267) to report on and enable/disable individual timing checks. **tcheck_status** prints in the Main window transcript the current status of all timing checks in the instance or a specific timing check specified with the optional **<tcheck>** argument.

Disabling a timing check's reporting prevents generation of associated violation messages. For Verilog modules this means ModelSim disables message reporting. For VHDL design units this means ModelSim sets the MsgOn parameter in a VITAL timing check procedure (TCP) to FALSE. Disabling a timing check's X generation removes a timing check's ability to affect the outputs of the simulation. For Verilog modules this means ModelSim toggles the timing check's notifier. For VHDL design units this means ModelSim sets the Xon parameter in a VITAL TCP to FALSE.

Syntax

```
tcheck_status
  [-lines] <instance> [<tcheck>]
```

Arguments

`-lines`
Specifies that the HDL source file and line numbers of the check(s) be displayed. Optional. Has no effect on VHDL instances. Note that line information may not always be available.

`<instance>`
Specifies the instance for which you want timing check status reported. Required.

`<tcheck>`
Specifies a specific timing check within the instance on which to report status. Optional. By default ModelSim reports all timing checks within the specified instance. You can specify either the integer that is assigned to each timing check (and reported via **tcheck_status**) or the actual timing check name enclosed in double quotes (see "Examples" below). Note that the integer number may change between library compiles.

Output

The output of the **tcheck_status** command looks as follows:

```
#<Number> <SDF_Description> [<src_line>] <MsgStat> <XStat>
```

Field	Description
<Number>	an integer that can be used as shorthand to specify the check in the **tcheck_status** or **tcheck_set** commands (as the **<tcheck>** argument); this number can change with compiler optimizations, and you can't assume it will stay the same between library compiles
<SDF_Description>	an SDF specification of the timing check including enclosing parentheses '( )'

Field	Description
<src_line>	the source file and line number for the timing check specification; output if you specify the **-lines** argument; the format of the item is `<source_file_name>:<line_number>`.
<MsgStat>	violation message reporting status indicator MsgON/MsgOFF - violation reporting is enabled/disabled and unchangeable MsgOn/MsgOff - violation reporting is enabled/disabled and modifiable
<XStat>	violation X generation status indicator XON/XOFF - X generation is enabled/disabled and unchangeable XOn/XOff - X generation is enabled/disabled and modifiable

Examples

```
tcheck_status top.y1.u2
```
Creates the following output:
```
  #0 ( WIDTH (negedge  CLK) ) MsgOn XOn
  #1 ( WIDTH (posedge  CLK) ) MsgOn XOn
  #2 ( SETUP (negedge  D) (posedge  CLK) ) MsgOFF XOFF
  #3 ( HOLD (posedge  CLK) (negedge  D) ) MsgOn XOff
```
```
tcheck_status -lines top.y1.u2 1
```
Creates the following output:
```
  #1 ( WIDTH (posedge  CLK) ) 'cell.v:224' MsgOn XOn
```

See also

tcheck_set (CR-267), *Chapter 17 - Standard Delay Format (SDF) Timing Annotation*

toggle add

The **toggle add** command enables collection of toggle statistics for the specified nodes. The allowed nodes are Verilog nets and registers and VHDL signals of type bit, bit_vector, std_logic, and std_logic_vector (other types are silently ignored).

You can also collect and view toggle statistics in the ModelSim GUI. See *Chapter 12 - Code Coverage* for details.

Syntax

```
toggle add
  [-full] [-in] [-inout] [-internal] [-out] [-ports] [-r] <node_name>
```

Returns

Command result	Return value
no signals are added and no signals are found to be already in the toggle set	Nothing added.
no signals are added and some signals are found to be already in the toggle set	0
some signals are added	the number of bits added

Arguments

`-full`
Enables extended mode toggle coverage, which tracks the following six transitions:

1) 1 or H --> 0 or L
2) 0 or L --> 1 or H
3) X or Z --> 1 or H
4) X or Z --> 0 or L
5) 1 or H --> X or Z
6) 0 or L --> X or Z

Optional. By default only transitions to 0 and 1 are counted.

`-in`
Enables toggle statistics collection on nodes of mode IN. Optional.

`-inout`
Enables toggle statistics collection on nodes of mode INOUT. Optional.

`-internal`
Enables toggle statistics collection on internal (non-port) items. Optional.

`-out`
Enables toggle statistics collection on nodes of mode OUT. Optional.

`-ports`
Enables toggle statistics collection on nodes of modes IN, OUT, or INOUT. Optional.

`-r`
Specifies that toggle statistics collection is enabled recursively into subregions. Optional. If omitted, toggle statistic collection is limited to the current region.

`<node_name>`
Enables toggle statistics collection for the named node(s). Required. Multiple names and wildcards are accepted.

Examples

```
toggle add /dut/data/a
```
Enables toggle statistics collection for signal */dut/data/a*.

```
toggle add {/dut/data_in[5]]}
```
Enables toggle statistics collection for bit 6 of bus */dut/data_in*. The curly braces must be added in order to escape the square brackets (’[]’).

See also

"Toggle coverage" (UM-437), **toggle report** (CR-275), **toggle reset** (CR-276)

toggle disable

The **toggle disable** command disables toggle statistics collection on the specified nodes. The command provides a method of implementing coverage exclusions for toggle coverage.

The command is intended to be used as follows:

1. Enable toggle statistics collection for all signals using the -cover t/x argument to **vcom** (CR-303) or **vlog** (CR-345).
2. Exclude certain signals by disabling them with the toggle disable command.

Syntax

```
toggle disable
  [-all] | [-in] [-out] [-inout] [-internal] [-ports] [-r] <node_name>
```

Arguments

`-all`
Disables toggle statistics collection for all nodes that have toggle checking enabled. Optional. Must be used alone without other arguments.

`-in`
Disables toggle statistics collection on nodes of mode IN. Optional.

`-out`
Disables toggle statistics collection on nodes of mode OUT. Optional.

`-inout`
Disables toggle statistics collection on nodes of mode INOUT. Optional.

`-internal`
Disables toggle statistics collection on internal (non-port) items. Optional.

`-ports`
Disables toggle statistics collection on nodes of modes IN, OUT, or INOUT. Optional.

`-r`
Specifies that toggle statistics collection is disabled recursively into subregions. Optional. If omitted, the disable is limited to the current region.

`<node_name>`
Disables toggle statistics collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

"Toggle coverage" (UM-437), **toggle add** (CR-271), **toggle enable** (CR-274)

toggle enable

The **toggle enable** command re-enables toggle statistics collection on nodes whose toggle coverage had previously been disabled via the toggle disable command.

Syntax

```
toggle enable
  [-all] | [-in] [-out] [-inout] [-internal] [-ports] [-r] <node_name>
```

Arguments

`-all`
Enables toggle statistics collection for all nodes that have toggle checking disabled. Optional. Must be used alone without other arguments.

`-in`
Enables toggle statistics collection on disabled nodes of mode IN. Optional.

`-out`
Enables toggle statistics collection on disabled nodes of mode OUT. Optional.

`-inout`
Enables toggle statistics collection on disabled nodes of mode INOUT. Optional.

`-internal`
Enables toggle statistics collection on disabled internal (non-port) items. Optional.

`-ports`
Enables toggle statistics collection on disabled nodes of modes IN, OUT, or INOUT. Optional.

`-r`
Specifies that toggle statistics collection is enabled recursively into subregions. Optional. If omitted, the enable is limited to the current region.

`<node_name>`
Enables toggle statistics collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

"Toggle coverage" (UM-437), **toggle disable** (CR-273)

toggle report

The **toggle report** command displays a list of all nodes that have not transitioned to both 0 and 1 at least once. Also displayed is a summary of the number of nodes checked, the number that toggled, the number that didn't toggle, and a percentage that toggled.

You can also collect and view toggle statistics in the ModelSim GUI. See *Chapter 12 - Code Coverage* for details.

The **toggle report** command is intended to be used as follows:

1. Enable statistics collection with the **toggle add** command (CR-271).
2. Run the simulation with the **run** command (CR-246).
3. Produce the report with the **toggle report** command.

Syntax

```
toggle report
  [-all] [-file <filename>] [<signal>...] [-summary]
```

Arguments

`-all`
Lists all nodes checked along with their individual transition to 0 and 1 counts. Optional.

`-file <filename>`
Specifies a file to which to write the report. By default the report is displayed in the Main window. Optional.

`<signal>...`
Specifies the name of a signal whose toggle statistics is to be displayed. Multiple signal names, separated by spaces, may be specified. Wildcards may be used.

`-summary`
Selects only the summary portion of the report. Optional.

See also

"Toggle coverage" (UM-437), **toggle add** (CR-271), **toggle reset** (CR-276)

toggle reset

The **toggle reset** command resets the toggle counts to zero for the specified nodes.

Syntax

```
toggle reset
  [-all] | [-in] [-out] [-inout] [-internal] [-ports] [-r] <node_name>
```

Arguments

`-all`
Resets toggle statistics collection for all nodes that have toggle checking enabled. Optional. Must be used alone without other arguments.

`-in`
Resets toggle statistics collection on nodes of mode IN. Optional.

`-out`
Resets toggle statistics collection on nodes of mode OUT. Optional.

`-inout`
Resets toggle statistics collection on nodes of mode INOUT. Optional.

`-internal`
Resets toggle statistics collection on internal (non-port) items. Optional.

`-ports`
Resets toggle statistics collection on nodes of modes IN, OUT, or INOUT. Optional.

`-r`
Specifies that toggle statistics collection is reset recursively into subregions. Optional. If omitted, the reset is limited to the current region.

`<node_name>`
Resets toggle statistics collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

"Toggle coverage" (UM-437), **toggle add** (CR-271), **toggle report** (CR-275)

transcribe

The **transcribe** command displays a command in the Main window, then executes the command. The transcribe command is normally used to direct commands to the Main window from an external event such as a menu pick or button selection. The **add button** (CR-52) and **add_menuitem** (CR-61) commands can utilize **transcribe**. Returns nothing.

Syntax

```
transcribe
  <command>
```

Arguments

`<command>`
Specifies the command to execute. Required.

Examples

`add button pwd {transcribe pwd} NoDisable`
Creates a button labeled "pwd" that invokes **transcribe** with the **pwd** Tcl command, and echoes the command and its results to the Main window. The button remains active during a run.

See also

add button (CR-52), **add_menuitem** (CR-61)

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 Main window as they are executed. Optional.

`off`
Specifies that commands in a macro file will not be echoed to the Main window as they are executed. Optional. The **transcribe** command (CR-277) can be used to force a command to be echoed.

`-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 (CR-233). Optional.

Examples

`transcript on`
Commands within a macro file will be echoed to the Main window as they are executed.

`transcript`
If issued immediately after the previous example, the message:

```
Macro transcripting is turned on.
```

appears in the Main window.

See also

echo (CR-161), **transcribe** (CR-277)

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

```
transcript file ""
```
Closes the current transcript file and stops writing data to the file. This is a method for reducing the size of your transcript.

```
transcript file ""
run 1 ms
transcript file transcript
run 1 ms
```
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.

See also

"Transcript" (UM-264)

tssi2mti

The **tssi2mti** command is used to convert a vector file in Fluence Technology (formerly TSSI) Standard Events Format into a sequence of **force** (CR-176) and **run** (CR-246) 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 Fluence Technology 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

```
tssi2mti trigger.def trigger.sef > trigger.do
```
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 exactly the same as the previous one, but uses the standard input instead.

See also

force (CR-176), **run** (CR-246), **write tssi** (CR-395)

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 (CR-258)

up

The **up** command searches for signal transitions or values in the specified List window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a signal takes on a particular value, or an expression of multiple signals evaluates to true. See the **down** command (CR-157) for related functionality.

The procedure for using **up** includes three steps: click on the desired signal; click on the desired starting location; issue the **up** command. (The **seetime** command (CR-257) can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

Syntax

```
up
  [-expr {<expression>}] [-falling] [-noglitch] [-rising]
  [-value <sig_value>] [-window <wname>] [<n>]
```

Arguments

`-expr {<expression>}`
The List window will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced List window. A signal may be specified either by its full path or by the shortcut label displayed in the List window.

See "GUI_expression_format" (CR-23) for the format of the expression. The expression must be placed within curly braces.

`-falling`
Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-noglitch`
Specifies that delta-width glitches are to be ignored. Optional.

`-rising`
Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-value <sig_value>`
Specifies a value of the signal to match. Optional. Must be specified in the same radix that the selected signal is displayed. Case is ignored, but otherwise must be an exact string match -- don't-care bits are not yet implemented.

`-window <wname>`
Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the **view** command (CR-320) to change the default window.

`<n>`
Specifies to find the nth match. Optional. If less than n are found, the number found is returned with a warning message, and the marker is positioned at the last match.

Examples

`up -noglitch -value FF23`
Finds the last time at which the selected vector transitions to FF23, ignoring glitches.

`up`
Goes to the previous transition on the selected signal.

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the "GUI_expression_format" (CR-23) and can be built with the aid of the "The GUI Expression Builder" (UM-395).

```
up -expr {clk'rising && (mystate == reading) && (/top/u3/addr ==
32'habcd1234)}
```
Searches up for an expression that evaluates to a boolean 1 when signal *clk* just changed from low to high and signal *mystate* is the enumeration reading and signal */top/u3/addr* is equal to the specified 32-bit hex constant.

`up -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}`
Searches up for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal */top/u3/addr* equals hex ac.

`up -expr {((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode == writing)}`
Searches up for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, clock just changed from low to high, and signal *mode* is enumeration writing.

See also

"GUI_expression_format" (CR-23), **view** (CR-320), **seetime** (CR-257), **down** (CR-157)

vcd add

The **vcd add** command adds the specified items to a VCD file. The allowed items are Verilog nets and variables and VHDL signals of type bit, bit_vector, std_logic, and std_logic_vector (other types are silently ignored).

All **vcd add** commands must be executed at the same simulation time. The specified items 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>]
  <item_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 (CR-296).

`<item_name>`
Specifies the Verilog or VHDL item to add to the VCD file. Required. Multiple items may be specified by separating names with spaces. Wildcards are accepted.

See also

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on 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 (CR-294) or "dump.vcd" if **vcd file** was not invoked.

See also

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files.

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 (CR-294) or "dump.vcd" if **vcd file** was not invoked.

See also

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files.

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] [-file <filename>] [-in] [-inout] [-out] [-unique]
  [-vcdstim ] <item_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.

`-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.

`-unique`
Generates unique VCD variable names for ports, even if those ports are connected to the same collapsed net. Optional.

`-vcdstim`
Ensure that the order that the port names appear in the VCD file matches the order that they are declared in the instance's module or entity declaration. Optional. See "Port order issues" (UM-564) for further information.

`<item_name>`
Specifies the Verilog or VHDL item to add to the VCD file. Required. Multiple items may be specified by separating names with spaces. Wildcards are accepted.

Examples

```
vcd dumpports -in -file counter.vcd /test_counter/dut/*
```
Creates a VCD file named *counter.vcd* of all IN ports in the region /test_counter/dut/.

```
vcd dumpports -file addern.vcd /testbench/uut/*
vsim -vcdstim addern.vcd addern -gn=8 -do "add wave /*; run 1000"
```
These two commands resimulate a design from a VCD file. See "Simulating with input values from a VCD file" (UM-562) for further details.

```
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
```
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. See "Replacing instances with output values from a VCD file" (UM-563) for more information.

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

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files.

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

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files.

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

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files.

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

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files.

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

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files.

vcd file

The **vcd file** command specifies the filename and state mapping for the VCD file created by a **vcd add** command (CR-284). The **vcd file** command is optional. If used, it must be issued before any **vcd add** commands.

Related Verilog task: $dumpfile

Note: vcd file is included for backward compatibility. Use the **vcd files** command (CR-296) if you want to use multiple VCD files during a single simulation.

Syntax

```
vcd file
  [-dumpports] [<filename>] [-map <mapping pairs>] [-nomap]
```

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 (CR-284) will accept only qualifying ports (silently ignoring all other specified items).

`<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.

`-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

VHDL	VCD	VHDL	VCD
1	1	-	x
Z	z		

See also

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files.
Verilog tasks are documented in the IEEE 1364 standard.

vcd files

The **vcd files** command specifies a filename and state mapping for a VCD file created by a **vcd add** command (CR-284). 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] <filename> [-map <mapping pairs>] [-nomap]
```

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.

`<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.

`-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		

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** (CR-301) and **vcd off** (CR-300) 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

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files. 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 (CR-294) or *dump.vcd* if **vcd file** was not invoked.

See also

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files. 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>`
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 the file designated by the **vcd file** command (CR-294) or *dump.vcd* if **vcd file** was not invoked.

See also

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files. 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 (CR-294) or *dump.vcd* if **vcd file** was not invoked.

See also

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files. 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** (CR-284) 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 (CR-294) or *dump.vcd* if **vcd file** was not invoked.

See also

See *Chapter 18 - Value Change Dump (VCD) Files* for more information on VCD files. 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 can be displayed in ModelSim using the **vsim -view** argument.

Syntax

```
vcd2wlf
  [-splitio] [-splitio_in_ext <extension>] [-splitio_out_ext <extension>]
  <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.

`<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.

vcom

The **vcom** command compiles VHDL source code into a specified working library (or to the **work** library by default).

This command may be invoked from within ModelSim or from the operating system command prompt. This command may also be invoked during simulation.

Compiled libraries are major-version dependent. For example you cannot use a library compiled with 5.7 in a simulation using 5.8 **vsim**. You would have to refresh the libraries using the **-refresh** argument to **vcom**. This is not true for minor versions (e.g., 5.7a libraries work in 5.7d).

Syntax

```
vcom
  [-87] [-93] [-2002] [+acc[=<spec>][+<entity>[(architecture)]]]
  [-check_synthesis] [-cover <stat>] [-debugVA] [-explicit]
  [-f <filename>] [-force_refresh] [-help] [-ignoredefaultbinding]
  [-ignorevitalerrors] [-just abcep] [-skip abcep] [-line <number>]
  [-lint] [-no1164] [-noaccel <package_name>] [-nocasestaticerror]
  [-nocheck] [-nocoverage] [-nodebug[=ports]] [-noindexcheck] [-nologo]
  [-nonstddriverinit] [-noothersstaticerror] [-nopsl] [-norangecheck]
  [-novital] [-novitalcheck] [-nowarn <number>]
  [-O0 | -O1 | -O4 | -O5] [-pedanticerrors]
  [-performdefaultbinding] [-pslfile <filename>] [-quiet] [-rangecheck]
  [-refresh] [-s] [-source] [-time] [-version]
  [-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 (see "Preference variables located in INI files" (UM-617)).

`-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 (see "Preference variables located in INI files" (UM-617)).

`-2002`
Specifies that the compiler is to support VHDL-2002. Optional. This is the default.

`+acc[=<spec>][+<entity>[(architecture)]]`
Enables access to design objects that would otherwise become unavailable due to optimizations. Optional. Note that using this option may reduce optimizations.

<spec> currently has only one choice:

v–Enable access to variables, constants, and aliases in processes that would otherwise be merged due to optimizations.

<entity> and (<architecture>) specify the design unit(s) in which to allow the access. If (<architecture>) is not specified, then all architectures of a given <entity> are enabled for access.

`-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 (UM-619) variable in the *modelsim.ini* file to set a permanent default.

`-cover <stat>`
Enables various coverage statistics collection. Optional.

<stat> is one or more of the following characters:

b–Collect branch statistics.

c–Collect condition statistics.

e–Collect expression statistics.

s–Collect statement statistics. Default.

t–Collect toggle statistics. Cannot be used if 'x' is specified.

x–Collect extended toggle statistics (see "Toggle coverage" (UM-437) for details). Cannot be used if 't' is specified.

By default only statement coverage is enabled when you invoke **vsim** with the **-coverage** option.

`-debugVA`
Prints a confirmation if a VITAL cell was optimized, or an explanation of why it was not, during VITAL level-1 acceleration. Optional.

`-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. Nesting of **-f** options is allowed.

The file syntax basically follows what you type on the command line with the exception that newline characters are ignored. Environment variable expansion (for example in a pathname) *does not* occur in **-f** files.

`-force_refresh`
Forces the refresh of a design unit. Optional. When the compiler refreshes a design unit, it checks each dependency to ensure its source has not been changed and recompiled. If a dependency has been changed and recompiled, the compiler will not refresh the dependent design unit (unless you use **-force_refresh**). To avoid potential errors or mismatches caused by the dependency recompilation, you should recompile the dependent design unit's source rather than use this switch.

`-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.

`-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.

`-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`
Enables a warning message if the result of the built-in concatenation operator ("&") is the actual for a subprogram formal parameter of an unconstrained array type. Optional.

`-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.

`-nocoverage`
Disables collection of statement coverage statistics, which is on by default. Optional.

`-nodebug[=ports]`
Hides the internal data of the compiled design unit. Optional. The design unit's source code, internal structure, signals, processes, and variables will not display in ModelSim's windows. In addition, none of the hidden objects may be accessed through the Dataflow window or with commands. This also means that you cannot set breakpoints or single step within this code. Don't compile with this switch until you're done debugging.

Note that this is not a speed switch like the "nodebug" option on many other products.

The optional **=ports** switch hides the ports for the lower levels of your design; it should only be used to compile the lower levels of the design. If you hide the ports of the top level you will not be able to simulate the design.

Design units or modules compiled with **-nodebug** can only instantiate design units or modules that are also compiled **-nodebug**.

`-noindexcheck`
Disables checking on indexing expressions to determine whether indices are within declared array bounds. Optional.

`-nologo`
Disables display of 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 connect 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 only does this in the specific cases where older versions used the actual signal's initial value.

`-noothersstaticerror`
Disables warnings that result from array aggregates with multiple choices having "others" clauses that are not locally static. Optional. If the **-pedanticerrors** switch is specified, this switch is ignored.

`-nopsl`
Instructs the compiler to ignore embedded PSL assertions. By default vcom parses any PSL assertion statements it finds in the specified files. See "Compiling and simulating assertions" (UM-506) for more information.

`-norangecheck`
Disables run time range checking. In some designs, this results in a 2X speed increase. Range checking is enabled by default or, once disabled, can be enabled using **-rangecheck**. See "Range and index checking" (UM-74) for additional 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 Variables or Signals windows.

`-novitalcheck`
Disables VITAL 2000 compliance checking if you are using VITAL 2.2b. Optional.

`-nowarn <number>`
Selectively disables an individual warning message. Optional. Multiple **-nowarn** switches are allowed. Warnings may be disabled for all compiles via the Main window **Options > Compile Options** menu command or the *modelsim.ini* file (see the "[vcom] VHDL compiler control variables" (UM-619)).

The warning message numbers are:

1 = unbound component
2 = process without a wait statement
3 = null range
4 = no space in time literal
5 = multiple drivers on unresolved signal
6 = VITAL compliance checks
7 = VITAL optimization messages
8 = lint checks
9 = signal value used in expression evaluated at elaboration
10 = VHDL-1993 constructs in VHDL-1987 code

`-O0 | -O1 | -O4 | -O5`
Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

Enable PE-level optimization with **-O1**. Optional. Note that changing from the default **-O4** to **-O1** may cause event order differences in your simulation.

Enable standard SE optimizations with **-O4**. Default. The main differences between **-O4** and **-O1** are that ModelSim attempts to improve memory management for vectors and accelerate VITAL Level 1 modules with **-O4**.

Enable maximum optimization with **-O5**. Optional. We recommend use of this switch with large sequential blocks only; other uses may significantly increase compile times. **-O5** attempts to optimize loops and prevents variable assignments in situations where a variable is assigned but is not actually used. Using the **+acc** argument to **vcom** will cancel this latter optimization.

`-pedanticerrors`
Forces ModelSim to error (rather than warn) on three conditions: 1) when a choice in a case statement is not a locally static expression; 2) when an array aggregate with multiple choices doesn’t have a locally static "others" choice; 3) when a generate statement without a BEGIN keyword exists between the declarative items and the concurrent statements. 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.

`-pslfile <filename>`
Identifies an external PSL assertion file to compile along with the VHDL source files. See "Compiling and simulating assertions" (UM-506) for more information.

`-quiet`
Disables 'Loading' messages. Optional.

`-rangecheck`
Enables run time range checking. Default. Range checking can be disabled using the **-norangecheck** argument. See "Range and index checking" (UM-74) for additional information.

`-refresh`
Regenerates a library image. Optional. By default, the work library is updated; use **-work <library>** to update a different library. See **vcom** "Examples" (CR-309) for more information.

`-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.

`-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.

`-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**.

`-version`
Returns the version of the compiler as used by the licensing tools, such as "Model Technology ModelSim SE vcom 5.5 Compiler 2000.01 Jan 29 2000".

`-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

```
vcom example.vhd
```
Compiles the VHDL source code contained in the file *example.vhd*.

```
vcom -87 o_units1.vhd o_units2.vhd
vcom -93 n_unit91.vhd n_unit92.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 -nodebug example.vhd
```
Hides the internal data of *example.vhd*. Models compiled with **-nodebug** cannot use any of the ModelSim debugging features; any subsequent user will not be able to see into the model.

```
vcom -nodebug=ports level3.vhd level2.vhd
vcom -nodebug top.vhd
```
The first line compiles and hides the internal data, plus the ports, of the lower-level design units, *level3.vhd* and *level2.vhd*. The second line compiles the top-level unit, *top.vhd*, without hiding the ports. It is important to compile the top level without **=ports** because top-level ports must be visible for simulation.

```
vcom -noaccel numeric_std example.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 -explicit 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.

```
ARITHMETIC.”=”(left, right)
```

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.

```
vcom -work mylib -refresh
```
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).

vcover convert

The **vcover convert** command converts a coverage file created in ModelSim 5.7 to a ModelSim 5.8 format. You can also use the command with the **-strip** or **-install** arguments to create a new data file with different levels of hierarchy. The command can be invoked within the ModelSim GUI or at the command line.

Syntax

```
vcover convert
  [-57] [-install <path>] [-log <filename>] [-strip <n>] <file> <outfile>
```

Arguments

`-57`
Converts the specified file to a 5.8 format. Optional.

`-install <path>`
Adds <path> as additional hierarchy on the front end of instance and signal names in the input file. Optional. This argument allows you to create a new coverage file with a different level of hierarchy.

`-log <filename>`
Specifies the file for outputting progress messages. Optional. By default these messages are output to *vcover.log*.

`-strip <n>`
Removes <n> levels of hierarchy from instance and signal names in the data files. Optional. This argument allows you to create a new coverage file with a different level of hierarchy.

`<file>`
Specifies the file you want to convert. Required.

`<outfile>`
Specifies the name of the new file you want to output. Required.

See also

vcover merge command (CR-311), **vcover stats** command (CR-313), *Chapter 12 - Code Coverage*

vcover merge

The **vcover merge** command merges multiple coverage reports without having to re-simulate the designs. It can be invoked within the ModelSim GUI or at the command line.

Syntax

```
vcover merge
  [-and] [-append] [-f <pathname>] [-install <path>] [-log <filename>]
  [-strip <n>] [-verbose] <outfile> <file1> <file2> <filen>...
```

Arguments

`-and`
Excludes statements in the output file *only* if they are excluded in all input files. Optional. By default a statement is excluded in the output merge file if the statement is excluded in any of the input files.

`-append`
Specifies that progress messages are to be appended to the current log file. Optional. By default a new log file is created each time you invoke the command.

`-f <pathname>`
Specifies a text file containing input filenames that you want to merge. Optional.

`-install <path>`
Adds <path> as additional hierarchy on the front end of instance and signal names in the data files. Optional. This argument allows you to merge coverage results from simulations that have different hierarchies.

`-log <filename>`
Specifies the file for outputting progress messages. Optional. By default these messages are output to *vcover.log*.

`-strip <n>`
Removes <n> levels of hierarchy from instance and signal names in the data files. Optional. This argument allows you to merge coverage results from simulations that have different hierarchies.

`-verbose`
Enables summary code coverage statistics to be computed and directed to the log file each time a file is merged into the base. The statistics are instance-based.

`<outfile>`
Specifies the name of the file that will contain the merged output. Required.

`<file1> <file2> <filen>...`
Specifies the file(s) you want to merge. Required. Multiple pathnames and wildcards are allowed.

Example

`vcover merge myresult myfile1 myfile2`
Merges code coverage statistics for *myfile1* and *myfile2* and writes them to *myresult*.

`vcover merge myresult2 /dut/*.cov`
Uses wildcards to merge all files with a *.cov* extension in a particular directory.

See also

vcover convert command (CR-310), **vcover stats** command (CR-313), *Chapter 12 - Code Coverage*

vcover stats

The **vcover stats** command computes and prints to *stdout* summary statistics for the specified file(s). It can be invoked within the ModelSim GUI or at the command line.

Syntax

```
vcover stats
  [-and] [-append] [-f <pathname>] [-incremental] [-install <path>]
  [-log <filename>] [-strip <n>] [-verbose] <file1> [<file2> <filen>...]
```

Arguments

`-and`
Excludes statements in the output merge file *only* if they are excluded in all input files. Optional. By default statements are excluded in the output file if the statement is excluded in any of the input files.

`-append`
Specifies that progress messages are to be appended to the current log file. Optional. By default a new log file is created each time you invoke the command.

`-f <pathname>`
Specifies a text file containing input filenames for which you want to produce statistics. Optional.

`-incremental`
Prints statistics for the specified files as if the files were merged one after the other in the listed order. Optional. For example, using this argument will cause vcover stats to print the statistics for <*file1*>, then any incremental coverage after merging <*file2*>, and then any incremental coverage after merging <*file3*> into the merge of <*file1*> and <*file2*>, and so forth. At the end it prints the total statistics for the full merge. The statistics are written to both *stdout* and *vcover.log*.

`-install <path>`
Adds <path> as additional hierarchy on the front end of instance and signal names in the data files. Optional. This argument allows you to merge coverage results from simulations that have different hierarchies.

`-log <filename>`
Specifies the file for outputting progress messages. Optional. By default these messages are output to *vcover.log*.

`-strip <n>`
Removes <n> levels of hierarchy from instance and signal names in the data files. Optional. This argument allows you to merge coverage results from simulations that have different hierarchies.

`-verbose`
Enables summary code coverage statistics to be computed and directed to the log file each time a file is merged into the base. The statistics are instance-based.

`<file1> [<file2> <filen>...]`
Specifies the file(s) for which you want summary statistics. Required. Multiple pathnames and wildcards are allowed.

See also

vcover convert command (CR-310), **vcover merge** command (CR-311), *Chapter 12 - Code Coverage*

vdel

The **vdel** command deletes a design unit from a specified library.

Syntax

```
vdel
  [-help] [-lib <library_name>] [-verbose]
  [-all | <design_unit> | [<arch_name>] | -allsystemc]
```

Arguments

`-allsystemc`
Deletes all SystemC modules in a design from the working directory. Optional.

`-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.

`-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.

`<design_unit>`
Specifies the entity, package, configuration, or module to be deleted. Required unless **-all** is used. This option is not supported for SystemC modules.

Examples

`vdel -all`
Deletes the **work** library.

`vdel -lib synopsys -all`
Deletes the **synopsys** library.

`vdel xor`
Deletes the entity named **xor** and all its architectures from the **work** library.

`vdel xor behavior`
Deletes the architecture named **behavior** of the entity **xor** from the **work** library.

`vdel base`
Deletes the package named **base** from the **work** library.

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. SystemC modules are listed with this command.

Syntax

```
vdir
  [-help] [-l] [-r] [-all] | [-lib <library_name>] [<design_unit>]
```

Arguments

`-help`
Displays the command's options and arguments. Optional.

`-l`
Prints the version of **vcom, vlog**, or **sccom** that each design unit was compiled under. Also prints the object-code version number that indicates which versions of **vcom/vlog/sccom** and ModelSim are compatible. This example was printed by **vdir -l** for the counter module in the **work** library:

```
# Library Vendor : Model Technology
# MODULE ram_tb
#     Verilog Version: DPV:j32Jc=Q?7<3><C;OK0
#     Version number: CRW2<UhheaW;LIL2_B5o31
#     Source modified time: 1064511064
#     Source file: ram_tb.v
#     Opcode format: 5.8 Beta 2; VLOG SE Object version 172
#     Optimized Verilog design root: 1
#     Language standard: 1
#     Source directory: C:\modelsim_examples\memory\vlog_memory
```

`-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. See "[Library] library path variables" (UM-617) for more information.

`-lib <library_name>`
Specifies the logical name or the pathname of the 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, and packages in the specified library are listed.

Examples

`vdir -lib design my_asic`
Lists the architectures associated with the entity named **my_asic** that reside in the HDL design library called **design**.

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
  [-all [-kind <tool]] [-fmt] [-ranges] <msgNum>...
```

Arguments

`-all [-kind <tool]`
Prints all error messges. Optional. If you specify -kind <tool>, it prints just those error messages associated with the specified tool.

`-fmt`
Prints the format string that is used in the actual error message. Optional.

`-ranges`
Prints the numeric ranges of error message numbers by tool. Optional.

`<msgNum>`
Specifies the message number of a ModelSim message. Required unless you specify the -all argument. The message number can be obtained from messages that have the format:

** <Level>: ([<Tool>-[<Group>-]]<MsgNum>) <FormattedMsg>

Example

Say you see the following message in the transcript:

```
** Error (vsim-3061) foo.v(22): Too many Verilog port connections.
```

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 opens a ModelSim window and brings that window to the front of the display. If multiple instances of a window exist, **view** will change the default window of that type to the specified window. Using the **-new** option, **view** will create an additional instance of the specified window type and set it to be the default window for that type.

Names for windows are generated as follows:

- The first window name (automatically created without using **-new**) has the same name as the window type.
- Additional window names created by **-new** append an integer to the window type, starting with 1.

To remove a window, use the **noview** command (CR-208).

The **view** command returns the name(s) of the viewed window(s).

Syntax

```
view
  [*] [-height <n>] [-icon] [-new] [-title {New Window Title}] [-width <n>]
  [-x <n>] [-y <n>] <window_type>...
```

Arguments

`*`
Specifies that all windows be opened. Optional.

`-height <n>`
Specifies the window height in pixels. Optional.

`-icon`
Toggles the view between window and icon. Optional.

`-new`
Creates a new instance of the window type specified with the **<window_type>** argument. Optional. New window names are created by appending an integer to the window type, starting with 1, then incrementing the integer.

`-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.

`-width <n>`
Specifies the window width in pixels. Optional.

`<window_type>...`
Specifies the ModelSim window type to view. Required. You do no need to type the full type (see examples below); implicit wildcards are accepted; multiple window types may be used. Available window types are:

```
assertions, dataflow, list, memory, process, signals, source, structure,
variables, wave
```

Also creates a new instance of the specified window type when used with the **-new** option. You may also specify the window(s) to view when multiple instances of that window type exist (e.g., wave2, structure1). This works only with ModelSim-generated window names, not with window titles specified with the **-title** argument.

`-x <n>`
Specifies the window upper-left-hand x-coordinate in pixels. Optional.

`-y <n>`
Specifies the window upper-left-hand y-coordinate in pixels. Optional.

Examples

```
view d
```
Opens the Dataflow window.

```
view m
```
Opens the Memory window.

```
view si pr
```
Opens the Signals and Process windows.

```
view s
```
Opens the Signals and Source windows.

```
view -title {My Wave Window} wave
```
Opens a new Wave window with My Wave Window as its title.

```
view wave
view -new wave
```
The first command creates a window named ’wave’. The second command creates a window named ‘wave1’. Its full Tk path is ‘.wave1’. Wave1 is now the default Wave window. Any **add wave** command (CR-64) would add items to wave1.

```
view wave
```
Changes the default Wave window back to ‘wave’.

```
add wave -win .wave1 mysig
```
Will override the default Wave window and add *mysig* to wave1.

See also

noview (CR-208)

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 any of the following: signals, functions, designs, 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 (CR-323), **virtual save** (CR-337), **virtual show** (CR-338), "Virtual Objects (User-defined buses, and more)" (UM-248)

virtual define

The **virtual define** command prints to the Main window 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 any of the following: signals, functions, designs, 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

`virtual define -kind explicits *`
Shows the definitions of all the virtuals you have explicitly created.

See also

virtual describe (CR-325), **virtual show** (CR-338), "Virtual Objects (User-defined buses, and more)" (UM-248)

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 any of the following: signals, functions, designs, 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

`virtual delete -kind explicits *`
Deletes all of the virtuals you have explicitly created.

See also

virtual signal (CR-339), **virtual function** (CR-327), "Virtual Objects (User-defined buses, and more)" (UM-248)

virtual describe

The **virtual describe** command prints to the Main window 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 any of the following: signals, functions, designs, 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

```
virtual describe -kind explicits *
```
Describes the data type of all virtuals you have explicitly created.

See also

virtual define (CR-323), **virtual show** (CR-338), "Virtual Objects (User-defined buses, and more)" (UM-248)

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

`vcd add [virtual expand myVirtualSignal]`
Adds 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. 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 (CR-339), "Virtual Objects (User-defined buses, and more)" (UM-248)

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" (CR-9) 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 Signals 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>] [-implicit] [-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.

`-implicit`
Used internally to create virtuals that are automatically saved with the List or Wave format. 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" (CR-23) 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

`virtual function { not /chip/section1/clk } clk_n`
Creates a signal */chip/section1/clk_n* that is the inverse of */chip/section1/clk*.

`virtual function -install /chip { (std_logic_vector) chip.vlog.rega } rega_slv`
Creates a std_logic_vector equivalent of a verilog register *rega* and installs it as */chip/rega_slv*.

`virtual function { /chip/addr[11:0] == 0xfab } addr_eq_fab`
Creates 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 { gate:/chip/siga XOR rtl:/chip/siga } siga_diff`
Creates 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 -delay {10 ns} {/top/signalA AND /top/signalB} myDelayAandB`
Creates a virtual signal consisting of the logical "AND" function of */top/signalA* with */top/signalB*, and delays it by 10 ns.

`virtual function { | (gate:/chip/outbus XOR rtl:/chip/outbus) } outbus_diff`
Creates 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.

Commands fully compatible with virtual functions

add list (CR-55)	add log /log (CR-187)	add wave (CR-64)
checkpoint (CR-99) and restore (CR-242)	delete (CR-151)	describe (CR-152) ("virtual describe" is a little faster)
down (CR-157) / up (CR-282)	examine (CR-167)	find (CR-172)
restart (CR-240)	left (CR-185) / right (CR-244)	search (CR-253)
searchlog (CR-255)	show (CR-260)	

Commands not currently compatible with virtual functions

check contention add (CR-90)	check contention config (CR-92)	check contention off (CR-93)
check float add (CR-94)	check float config (CR-95)	check float off (CR-96)
check stable on (CR-98)	check stable off (CR-97)	drivers (CR-159)
force (CR-176)	noforce (CR-204)	power add (CR-216)
power report (CR-217)	power reset (CR-218)	toggle add (CR-271)
toggle reset (CR-276)	toggle report (CR-275)	vcd add (CR-284)
when (CR-375)		

See also

virtual count (CR-322)	virtual define (CR-323)	virtual delete (CR-324)
virtual describe (CR-325)	virtual expand (CR-326)	virtual hide (CR-330)
virtual log (CR-331)	virtual nohide (CR-333)	virtual nolog (CR-334)
virtual region (CR-336)	virtual save (CR-337)	virtual show (CR-338)
virtual signal (CR-339)	virtual type (CR-342)	Virtual Objects (User-defined buses, and more) (UM-248)

virtual hide

The **virtual hide** command sets a flag in the specified real or virtual signals, so those signals do not appear in the Signals 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 any of the following: signals, functions, designs, 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 (CR-333), "Virtual Objects (User-defined buses, and more)" (UM-248)

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 any of the following: signals, functions, designs, 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) items 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 (CR-334), "Virtual Objects (User-defined buses, and more)" (UM-248)

virtual nohide

The **virtual nohide** command reverses the effect of a **virtual hide** command. It resets the flag in the specified real or virtual signals, so those signals reappear in the Signals window.

Syntax

```
virtual nohide
  [-kind <kind>]|[-region <path>] <pattern>
```

Arguments

`-kind <kind>`
Specifies a subset of virtuals to look at. Optional. <kind> can be any of the following: signals, functions, designs, 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 (CR-330), "Virtual Objects (User-defined buses, and more)" (UM-248)

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 any of the following: signals, functions, designs, 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) items 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 (CR-331), "Virtual Objects (User-defined buses, and more)" (UM-248)

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 (CR-327), **virtual signal** (CR-339), "Virtual Objects (User-defined buses, and more)" (UM-248)

Note: Virtual regions cannot be used in the **when** (CR-375) 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 any of the following: signals, functions, designs, 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 (CR-322), "Virtual Objects (User-defined buses, and more)" (UM-248)

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 any of the following: signals, functions, designs, implicits, and explicits. Unique abbreviations are accepted.

See also

virtual define (CR-323), **virtual describe** (CR-325), "Virtual Objects (User-defined buses, and more)" (UM-248)

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" (CR-28) for more details on syntax.

Syntax

```
virtual signal
  [-env <path>] [-install <path>] [-implicit] [-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.

`-implicit`
Used internally to create virtuals that are automatically saved with the List or Wave format. 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" (CR-9).

`<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

```
virtual signal -env sim:/chip/alu { (concat_range (4 downto 0))(a_04 & a_03
& a_02 & a_01 & a_00) } a
```
Reconstructs 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 [4:0])&{a_04, a_03, a_02,
a_01, a_00} } a
```
Reconstructs a bus *sim:chip.alu.a[4:0]*, using Verilog notation. Note that the concatenation notation starts with "&{" rather than "{".

```
virtual signal -install sim:/testbench { /chipa/alu/a(19 downto 13) &
/chipa/decode/inst & /chipa/mode } stuff
```
Creates 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 -delay {10 ps} {/top/signalA} myDelayedSignalA
```
Creates a virtual signal that is the same as */top/signalA* except it is delayed by 10 ps.

```
virtual signal { chip.instruction[23:21] } address_mode
```
Creates a three-bit signal, *chip.address_mode*, as an alias to the specified bits.

```
virtual signal {a & b & c & 3'b000} myextendedbus
```
Concatenates signals *a*, *b*, and *c* with the literal constant ’000’.

```
virtual signal {num & "000"} fullbus
add wave -unsigned fullbus
```
Adds three missing bits to the bus *num*, creates a virtual signal *fullbus*, and then adds that signal to the Wave window.

```
virtual signal { num31 & num30 & num29 & ... & num4 & num3 & "000" } fullbus
add wave -unsigned fullbus
```
Reconstructs 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 {(aold == anew) & (bold == bnew)} myequalityvector
```
Creates 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 {(concat_reverse)(bus1 & bus2[7:4])} newbus
```
Creates signal *newbus* that is a concatenation of bus1 (bit-reveresed) 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" (CR-29) for further details.

Commands fully compatible with virtual signals

add list (CR-55)	add log / log (CR-187)	add wave (CR-64)
checkpoint (CR-99) and restore (CR-242)	delete (CR-151)	describe (CR-152) ("virtual describe" is a little faster)
down (CR-157) / up (CR-282)	examine (CR-167)	find (CR-172)
force (CR-176)/noforce (CR-204)	restart (CR-240)	left (CR-185) / right (CR-244)
search (CR-253)	searchlog (CR-255)	show (CR-260)

Commands compatible with virtual signals using [virtual expand <signal>]

check contention add (CR-90)	check contention config (CR-92)	check contention off (CR-93)
check float add (CR-94)	check float config (CR-95)	check float off (CR-96)
check stable on (CR-98)	check stable off (CR-97)	drivers (CR-159)
power add (CR-216)	power report (CR-217)	power reset (CR-218)
toggle add (CR-271)	toggle reset (CR-276)	toggle report (CR-275)
vcd add (CR-284)		

Commands not currently compatible with virtual signals

when (CR-375)

See also

virtual count (CR-322)	virtual define (CR-323)	virtual delete (CR-324)
virtual describe (CR-325)	virtual expand (CR-326)	virtual function (CR-327)
virtual hide (CR-330)	virtual log (CR-331)	virtual nohide (CR-333)
virtual nolog (CR-334)	virtual region (CR-336)	virtual save (CR-337)
virtual show (CR-338)	virtual type (CR-342)	Virtual Objects (User-defined buses, and more) (UM-248)

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** (CR-375) 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

```
virtual type {state0 state1 state2 state3} mystateType
virtual function {(mystateType)mysignal} myConvertedSignal
add wave myConvertedSignal
```

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 Signals window, the string "state0" will appear when *mysignal* == 0, "state1" when *mysignal* == 1, "state2" when *mysignal* == 2, etc.

```
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
```

Uses 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 -delete mystateType
```

Deletes the virtual type "mystateType".

```
virtual type {{0x01-- add}{0x02-- sub}{default bad}} mydecodetype
```

Creates a virtual type that includes "don’t-cares" (the ’-’ character).

```
virtual type {{0x0100 0xff add}{0x0200 0xff sub}{default bad}} mydecodetype
```

Creates 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.

See also

virtual function (CR-327), "Virtual Objects (User-defined buses, and more)" (UM-248)

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>]] [-help] [-dos | -short | -unix | -long]
  <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. See "Archives" (UM-55) 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.

`-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** (CR-355) 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 ModelSim SE.

`-long`
Interchangeable with the **-unix** argument. Optional.

`<name>`
Specifies the pathname or archive name of the library to be created. Required.

Examples

`vlib design`
Creates the design library *design*. You can define a logical name for the library using the **vmap** command (CR-356) or by adding a line to the library section of the *modelsim.ini* file that is located in the same directory.

`vlib -archive -compact .3 uut`
Creates 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.

vlog

The **vlog** command compiles Verilog source code into a specified working library (or to the **work** library by default).

vlog 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. For example you cannot use a library compiled with 5.7 in a simulation using 5.8 **vsim**. You would have to refresh the libraries using the **-refresh** argument to **vlog**. This is not true for minor versions (e.g., 5.7a libraries work in 5.7d).

Syntax

```
vlog
  [-93] [-help] [-compat] [-compile_uselibs[=<directory_name>]]
  [-cover <stat>] [-debugCellOpt] [+define+<macro_name>[=<macro_text>]]
  [+delay_mode_distributed] [+delay_mode_path] [+delay_mode_unit]
  [+delay_mode_zero] [-f <filename>]
  [-fast[=<secondary_name>] [+acc[=<spec>] [+<module>[.]]]] [-forcecode]
  [-hazards] [+incdir+<directory>] [-incr] [-instantiateReadOnly]
  [-keep_delta] [-L <libname>] [-Lf <libname>] [+libext+<suffix>]
  [-libmap <pathname>] [-libmap_verbose] [+librescan] [-line <number>]
  [-lint] [+maxdelays] [+mindelays] [+nocheckALL] [+nocheckCLUP]
  [+nocheckDELAY] [+nocheckDNET] [+nocheckOPRD] [+nocheckSUDP]
  [-nocoverage][-nodebug[=ports | =pli]] [-noincr] [+nolibcell] [-nologo]
  [+nospecify] [+notimingchecks] [+nowarn<CODE>]
  [-O0 | -O1 | -O4 | -O5] [+opt+[<lib>.]<module>] [+protect] [-quiet]
  [-R [<simargs>]] [-refresh] [-source] [-sv] [-time] [+typdelays] [-u]
  [-v <library_file>] [-version] [-vlog95compat] [-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.

`-help`
Displays the command's options and arguments. 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. See "Event ordering in Verilog designs" (UM-119) for additional information.

`-compile_uselibs[=<directory_name>]`
Locates source files specified in a **`uselib** directive (see "Verilog-XL `uselib compiler directive" (UM-114)), 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.

`-cover <stat>`
Specifies type(s) of coverage statistics to collect. Optional.

<stat> is one or more of the following characters:

b–Collect branch statistics.

c–Collect condition statistics.

e–Collect expression statistics.

s–Collect statement statistics.

t–Collect toggle statistics. Cannot be used if 'x' is specified.

x–Collect extended toggle statistics (see "Toggle coverage" (UM-437) for details). Cannot be used if 't' is specified.

By default only statement coverage is enabled when you invoke **vsim** with the **-coverage** option.

`-debugCellOpt`
Produces Main window transcript output that identifies why certain cells within the design were not optimized. Used only when compiling gate-level Verilog libraries with **-fast** (see below). Optional.

`+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. See "Delay modes" (UM-142) for details.

`+delay_mode_path`
Sets distributed delays to zero in favor of using path delays. Optional. See "Delay modes" (UM-142) for details.

`+delay_mode_unit`
Sets path delays to zero and non-zero distributed delays to one time unit. Optional. See "Delay modes" (UM-142) for details.

`+delay_mode_zero`
Sets path delays and distributed delays to zero. Optional. See "Delay modes" (UM-142) for details.

`-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.

The file syntax basically follows what you type on the command line with the exception that newline characters are ignored. Environment variable expansion (for example in a pathname) *does not* occur in **-f** files.

`-fast[=<secondary_name>] [+acc[=<spec>] [+<module>[.]]]`
Increases simulation speed by allowing parameter propagation and global optimizations. Optional. To use this parameter, you must compile the source code for your entire design in a single invocation of the compiler. The following options are available:

`=<secondary_name>`
Allows you to specify a different secondary name for the optimized code. The compiler automatically assigns a secondary name to distinguish optimized code from un-optimized code that may exist in the same library. The default secondary name for optimized code is "fast"; the default secondary name for un-optimized code is "verilog".

`+acc[=<spec>][+<module>[.]]`
Allows you to maintain design object visibility. Note that using this option may reduce simulation speed.

<spec> is one or more of the following characters:

b–Enable access to bits of vector nets. This is necessary for PLI applications that require handles to individual bits of vector nets. Also, some user interface commands require this access if you need to operate on net bits.

c–Enable access to library cells. By default any Verilog module containing a non-empty specify block may be optimized, and debug and PLI access may be limited. This option keeps module cell visibility.

l–Enable access to line number directives and process names.

n–Enable access to nets.

p–Enable access to ports. This disables the module inlining optimization, and is necessary only if you have PLI applications that require access to port handles.

r–Enable access to registers (including memories, integer, time, and real types).

s–Enable access to system tasks.

t–Enable access to tasks and functions.

If <spec> is omitted, access is enabled for all objects.

<module> is a module name, optionally followed by "." to indicate all children of the module. Multiple modules are allowed, with each separated by a "+". If no modules are specified, then all modules are affected.

Please see additional discussion about **-fast** in "Compiling for faster performance" (UM-127). Also, see **+opt** argument below.

`-forcecode`
Forces code generation for optimized inlined modules when using the **-fast** switch. Optional. Use only in conjunction with the **-fast** switch. By default, code is not generated for inlined modules when the **-fast** switch is used.

`-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** (CR-357). See "Hazard detection" (UM-122) for more details.

Important: Enabling **-hazards** implicitly enables the **-compat** argument. As a result, using this argument may affect your simulation results.

`+incdir+<directory>`
Specifies directories to search for files included with **`include** compiler directives. Optional. By default, the current directory is searched first and then the directories specified by the **+incdir** options in the order they appear on the command line. You may specify multiple **+incdir** options as well as multiple directories separated by "+" in a single **+incdir** option.

`-incr`
Performs an incremental compile. Optional. 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. May be used with **-fast**.

`-instantiateReadOnly`
Enables a -fast optimized design to instantiate modules or UDPs from a work library that has read-only permission. The instantiations will not be in-lined or further optimized. Recommended usage is to always have write access to the work library.

`-keep_delta`
Disables optimizations that remove delta delays. Optional.

Delta delays result from zero delay events. Those events are normally processed in the next iteration or "delta" of the current timestep. **-fast** and **+opt** implement optimizations that can remove delta delays and process an event earlier.

`-L <libname>`
Searches the specified resource library for precompiled modules. Optional.

This argument can be used in tandem with **-fast** (see above) when you need to optimize pre-compiled modules for which you don't have source code. The library search options you specify here must also be specified when you run the **vsim** command (CR-357).

`-Lf <libname>`
Same as **-L** but the specified library is searched before any **'uselib** directives. (See "Library usage" (UM-111) and "Verilog-XL `uselib compiler directive" (UM-114) for more information). Optional.

`+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 three 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. The warnings are reported as WARNING[8]. Can also be enabled using the Show_Lint variable in the *modelsim.ini* file.

`+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 (except when compiling with **-fast**, in which case the compiler actually throws away the delay values that aren't needed).

`+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 (except when compiling with **-fast**, in which case the compiler actually throws away the delay values that aren't needed).

`+nocheckALL`
Enables all **+nocheck** arguments described below. Optional. Argument has an effect only when compiling gate-level cell libraries with **-fast** (see above). The **+nocheck** switches increase the optimizations of **-fast**.

`+nocheckCLUP`
Allows connectivity loops in a cell to be optimized. Optional. Argument has an effect only when compiling gate-level cell libraries with **-fast** (see above).

`+nocheckDELAY`
When used in conjunction with **+delay_mode_path** (see above), allows inlined Verilog modules with distributed delays and no path delays to be optimized. Optional. Argument has an effect only when compiling gate-level cell libraries with **-fast** (see above).

`+nocheckDNET`
Allows both the port and the delayed port (created for negative setup/hold) to be used in the functional section of the cell. Optional. Argument has an effect only when compiling gate-level cell libraries with **-fast** (see above).

`+nocheckOPRD`
Allows an output port to be read internally by the cell. Optional. Argument has an effect only when compiling gate-level cell libraries with **-fast** (see above). Note that if the value read is the only value contributed to the output by the cell, and if there's a driver on the net outside the cell, the value read will not reflect the resolved value.

`+nocheckSUDP`
Allows a sequential UDP to drive another sequential UDP. Optional. Argument has an effect only when compiling gate-level cell libraries with **-fast** (see above).

`-nocoverage`
Disables collection of statement coverage statistics, which are on by default. Optional.

`-nodebug[=ports | =pli]`
Hides the internal data of the compiled design unit. Optional. The design unit's source code, internal structure, registers, nets, etc. will not display in ModelSim's windows. In addition, none of the hidden objects may be accessed through the Dataflow window or with commands. This also means that you cannot set breakpoints or single step within this code. Don't compile with this switch until you're done debugging.

Note that this is not a speed switch like the "nodebug" option on many other products. Use the **-fast** switch to increase simulation speed.

The optional **=ports** switch hides the ports for the lower levels of your design; it should be used only to compile the lower levels of the design. If you hide the ports of the top level you will not be able to simulate the design.

The optional **=pli** switch prevents the use of pli functions to interrogate individual modules for information; this switch may be used at any level of the design. Combine both switches with **=ports+pli** or **=pli+ports**.

-nodebug encrypts entire files. The **`protect** compiler directive allows you to encrypt regions within a file. See "ModelSim compiler directives" (UM-152) for details.

`-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.

`+notimingchecks`
Removes all timing check entries from the design as it is parsed. Optional. To disable checks on individual instances, use the **tcheck_set** command (CR-267).

`+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**.

`-O0 | -O1 | -O4 | -O5`

Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

Enable PE-level optimization with **-O1**. Optional.

Enable standard SE optimizations with **-O4**. Default.

Enable maximum optimization with **-O5**. Optional. **-O5** attempts to optimize loops and prevents variable assignments in situations where a variable is assigned but is not actually used. Using the **+acc** argument to **vlog** will cancel this latter optimization.

Use caution with the **-O5** argument. We recommend use of this argument with large sequential blocks only; other uses may significantly increase compile times. Also, before using **-O5** with **-fast** (described above), try using both switches independently to make sure the optimized design behaves the same as the original version.

`+opt+[<lib>.]<module>`

Generates optimized code for designs that have been previously compiled un-optimized (without the **-fast** option; see above). Optional. The <module> specification is the name of the top-level design module, and <lib>, which is optional, is the library in which it resides. By default, the top-level module is searched for in the work library. If the design has multiple top-level modules, then provide the names in a list separated by plus signs. For example,

```
vlog +opt+testbench+globals
```

Any options that are appropriate with **-fast** are also appropriate with **+opt**. Specifically, use the **+acc** option to enable PLI access, and use the **-L** and **-Lf** options to specify the libraries to be searched.

Please see additional discussion about **+opt** and **-fast** in "Compiling for faster performance" (UM-127).

`+protect`

Enables **`protect ... `endprotect** directives. Optional. See "ModelSim compiler directives" (UM-152) for more information.

`-quiet`

Disables 'Loading' messages. Optional.

`-R [<simargs>]`

Instructs the compiler to invoke **vsim** (CR-357) 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; use **-work <library_name>** to update a different library. See **vlog** examples for more information.

`-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.

`-sv`
Enables System Verilog keywords. Optional. By default ModelSim follows the rules of IEEE Std 1364-2001 and ignores System Verilog keywords. If a source file has a ".sv" extension, ModelSim will automatically parse System Verilog keywords.

`-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**.

`+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. See "Verilog-XL compatible compiler arguments" (UM-113) 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, such as "Model Technology ModelSim SE vlog 5.5 Compiler 2000.01 Jan 28 2000".

`-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 (UM-618) variable in the *modelsim.ini* file to set a permanent default.

`-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. See "Verilog-XL compatible compiler arguments" (UM-113) 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.

Important: 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

```
vlog example.vlg
```
Compiles the Verilog source code contained in the file *example.vlg*.

```
vlog -nodebug example.v
```
Hides the internal data of *example.v*. Models compiled with **-nodebug** cannot use any of the ModelSim debugging features; any subsequent user will not be able to see into the model.

```
vlog -nodebug=ports level3.v level2.v
vlog -nodebug top.v
```
The first line compiles and hides the internal data, plus the ports, of the lower-level design units, *level3.v* and *level2.v*. The second line compiles the top-level unit, *top.v*, without hiding the ports. It is important to compile the top level without **=ports** because top-level ports must be visible for simulation.

```
vlog -nodebug=ports+pli level3.v level2.v
vlog -nodebug=pli top.v
```
The first command hides the internal data, and ports of the design units, *level3.v* and *level2.v*. In addition it prevents the use of PLI functions to interrogate the compiled modules for information (either **=ports+pli** or **=pli+ports** works fine for this command). The second line compiles the top-level unit without hiding the ports but restricts the use of PLI functions as well.

Note that the **=pli** switch may be used at any level of the design but **=ports** should only be used on lower levels since you can't simulate without visible top-level ports.

```
vlog -L work -L libA -L libB top.v
```
This command demonstrates how to compile hierarchical modules organized into separate libraries that have sub-module names that overlap among the libraries. Assume you have a top-level module *top* that instantiates module *modA* from library *libA* and module *modB* from library *libB*. Furthermore, *modA* and *modB* both instantiate modules named *cellA*, but the definition of *cellA* compiled into *libA* is different from that compiled into *libB*. In this case, you can't just specify **-L libA - L libB** because instantiations of *cellA* from *modB* resolve to the *libA* version of *cellA*. See "Library usage" (UM-111) for further information.

`vlog -fast cpu_rtl.v`
Compiles all modules in *cpu_rtl.v* using global optimizations. Assuming your top-level module is named testbench, you would simulate the design as follows:

```
vsim -c testbench
```

`vlog -fast=opt1 cpu_rtl.v`
Compiles all modules in *cpu_rtl.v* using global optimizations, and assigns the secondary name *opt1* to the optimized modules.

`vlog -fast +acc+foo cpu_rtl.v`
Compiles modules in *cpu_rtl.v* using global optimizations, but preserves visibility on module *foo*.

`vlog -fast +acc+foo. cpu_rtl.v`
Compiles modules in *cpu_rtl.v* using global optimizations, but preserves visibility on module *foo* and all its children.

`vlog -fast +acc=rn cpu_rtl.v`
Compiles *cpu_rtl.v* using global optimizations, but enables net and register access in all modules in the design.

Note: Please see additional discussion about **-fast** in "Compiling for faster performance" (UM-127) in the Verilog simulation chapter.

`vlog top.v -v und1`
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 +libext+.v+.u -y vlog_lib`
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 -work mylib -refresh`
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).

If your library contains VHDL design units be sure to regenerate the library with the **vcom** command (CR-303) using the **-refresh** option as well. See "Regenerating your design libraries" (UM-63) for more information.

`vlog module1.v -u -O0 -incr`
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.

vmake

The **vmake** utility allows you to use a UNIX or Windows MAKE program to maintain libraries. You run **vmake** on a compiled design library, and the utility outputs a makefile. You can then run the makefile with a version of MAKE (not supplied with ModelSim) to reconstruct the library. A MAKE program is included with Microsoft Visual C/C++, as well as many other program development environments.

After running the **vmake** utility, MAKE recompiles only the design units (and their dependencies) that have changed. You run **vmake** only once; then you can simply run MAKE to rebuild your design. If you add new design units or delete old ones, you should re-run **vmake** to generate a new makefile.

The **vmake** utility ignores library objects compiled with **-nodebug**. Also, the **vmake** utility is not supported for use with SystemC.

This command must be invoked from either the UNIX or the Windows/DOS prompt.

Syntax

```
vmake
  [-fullsrcpath] [-help] [<library_name>] [><makefile>]
```

Arguments

`-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.

`<library_name>`
Specifies the library name; if none is specified, then **work** is assumed. Optional.

`><makefile>`
Specifies the makefile name. Optional.

Examples

To produce a makefile for the work library:

```
vmake >makefile
```

You can also run **vmake** on libraries other than **work**:

```
vmake mylib >mylib.mak
```

To rebuild **mylib,** specify its makefile when you run MAKE:

```
make -f mylib.mak
```

vmap

The **vmap** command defines a mapping between a logical library name and a directory by modifying the *modelsim.ini* file. With no arguments, **vmap** reads the appropriate *modelsim.ini* file(s) and prints 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, or to view the results of a previous simulation run (when invoked with the **-view** switch). You can specify a VHDL configuration or an entity/architecture pair, or a Verilog module or configuration for simulation. If you specify a VHDL configuration, it is invalid to specify an architecture. During elaboration **vsim** determines if the source has been modified since the last compile.

To **manually interrupt design elaboration** use the Break key or <Ctrl-c> from a shell.

You may also invoke the **vsim** command from the command line within ModelSim with most of the options shown below (all except the **vsim -c** and **-restore** options).

Syntax

```
vsim
  [-assertfile <filename>] [-c] [-csupv2] [-compress_elab] [-coverage]
  [-do “<command_string>” | <macro_file_name>] [+dumpports+direction]
  [+dumpports+unique] [-elab <filename>] [-elab_cont <filename>]
  [-elab_defer_fli] [-f <filename>]
  [-filemap_elab <HDLfilename>=<NEWfilename>]
  [-g<Name>=<Value> ...] [-G<Name>=<Value> ...] [-gui]
  [-help] [-i] [-installcolormap] [-keeploaded] [-keeploadedrestart]
  [-keepstdout] [-l <filename>] [-lib <libname>] [<license_option>]
  [-load_elab <filename>] [-multisource_delay min | max | latest]
  [+multisource_int_delays] [-nocompress] [+no_notifier] [+no_tchk_msg]
  [+notimingchecks] [+pulse_int_e/<percent>] [+pulse_int_r/<percent>]
  [-quiet] [-restore <filename>]
  [-sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=]<sdf_filename>]
  [-sdfmaxerrors <n>] [-sdfnoerror] [-sdfnowarn] [+sdf_verbose]
  [-t [<multiplier>]<time_unit>] [-tag <string>] [-title <title>]
  [-trace_foreign <int>] [+transport_int_delays]
  [-vcdstim [<instance>=]<filename>] [-version]
  [-view [<dataset_name>=]<WLF_filename>] [-wlf <filename>] [-wlfcompress]
  [-wlfopt] [-wlfnocompress] [-wlfnoopt] [-wlfslim <size>]
  [-wlftlim <duration>]

  [-absentisempty] [-foreign <attribute>] [-nocollapse] [-nofileshare]
  [-noglitch] [+no_glitch_msg] [-nopsl] [-std_input <filename>]
  [-std_output <filename>] [-strictvital] [-vcdread <filename>]
  [-vital2.2b]

  [+alt_path_delays] [+delayed_timing_checks]
  [-extend_tcheck_data_limit <percent>]
  [-extend_tcheck_ref_limit <percent>]
  [-hazards] [+int_delays] [-L <library_name> ...] [-Lf <library_name> ...]
  [+maxdelays] [+mindelays] [+no_cancelled_e_msg] [+no_neg_tchk]
  [+no_notifier] [+no_path_edge] [+no_pulse_msg] [+no_show_cancelled_e]
  [+no_tchk_msg] [+nosdferror] [+nosdfwarn] [+nospecify] [+nowarn<CODE>]
  [+ntc_warn] [-pli "<object list>"] [+<plusarg>]
  [+pulse_e/<percent>] [+pulse_e_style_ondetect] [+pulse_e_style_onevent]
  [+pulse_r/<percent>] [+sdf_nocheck_celltype] [+show_cancelled_e]
  [+transport_path_delays] [+typdelays] [-v2k_int_delays]

  [<library_name>.<design_unit>]
```

VSIM arguments are grouped alphabetically by language:

- Arguments, VHDL and Verilog (CR-358)
- Arguments, VHDL (CR-366)

- Arguments, Verilog (CR-367)
- Arguments, object (CR-371)

Arguments, VHDL and Verilog

`-assertfile <filename>`
Designates an alternative file for recording assertion messages. Optional. By default assertion messages are output to the file specified by the TranscriptFile variable in the *modelsim.ini* file (see "Creating a transcript file" (UM-628)).

`-c`
Specifies that the simulator is to be run in command line mode. Optional. Also see "ModelSim modes of operation" (UM-23) for more information.

`-csupv2`
Instructs **vsim** to use */usr/lib/libCsup_v2.sl* for shared object loading. Optional. Use this argument only on HP-UX 11.00 when you have compiled FLI/PLI/VPI C++ code with aCC's -AA option. This option may also be specified with the UseCsupV2 (UM-625) variable in the *modelsim.ini* file.

`-compress_elab`
Compresses an elaboration file when it is created. Optional. See "Simulating with an elaboration file" (UM-136) for more information.

`-coverage`
Enables Code Coverage statistics collection during simulation. Optional. See *Chapter 12 - Code Coverage* for more information.

`-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+direction`
Modifies the format of extended VCD files to contain direction information. Optional.

`+dumpports+unique`
Generates unique VCD variable names for ports in a VCD file, even if those ports are connected to the same collapsed net. Optional.

`-elab <filename>`
Creates an elaboration file for use with **-load_elab**. Optional. See "Simulating with an elaboration file" (UM-136) for more information.

`-elab_cont <filename>`
Creates an elaboration file for use with **-load_elab** and then continues the simulation. Optional.

`-elab_defer_fli`
Defers the initialization of FLI models until the load of the elaboration file. Use this argument along with **-elab** to create elaboration files for designs with FLI models that don't support checkpoint/restore. Note that FLI models sensitive to design load ordering may still not work correctly even if you use this argument. See "Simulating with an elaboration file" (UM-136) for more information.

`-f <filename>`
Specifies a file with more command line arguments. Optional. Allows complex argument strings to be reused without retyping. Environment variable expansion (for example in a pathname) *does not* occur in -f files.

`-filemap_elab <HDLfilename>=<NEWfilename>`
Defines a file mapping during **-load_elab** that lets you change the stimulus. Optional. See "Simulating with an elaboration file" (UM-136) 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. VHDL type mismatches will cause the specification to be ignored (including no error messages).

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 a forward tick 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"}
```

Note: When you compile Verilog code with **-fast** (see vlog (CR-345)), all parameter values are set at compile time. Therefore, the **-g** option has no effect on these parameters.

`-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>.

`-gui`
Starts the ModelSim GUI without loading a design. 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 "Main window" (UM-262) transcript to <filename>. Optional. Default is *transcript*. Can also be specified via the *modelsim.ini* (see "Creating a transcript file" (UM-628)) file or the *modelsim.tcl* preference file.

`-lib <libname>`
Specifies the default working library where **vsim** will look for the design unit(s). Optional. Default is "work".

`<license_option>`
Restricts the search of the license manager. Optional. Use one of the following options.

<license_option>	Description
-lic_nomgc	exclude any MGC licenses from the search
-lic_nomti	exclude any MTI licenses from the search
-lic_noqueue	do not wait in queue when license is unavailable
-lic_plus	checks out ModelSim SE/PLUS (VHDL and Verilog) license immediately after invocation
-lic_vhdl	checks out ModelSim SE/VHDL license immediately after invocation
-lic_vlog	checks out ModelSim SE/VLOG license immediately after invocation
-lic_viewsim	accepts a simulator license rather than being queued for a viewer license

The options may also be specified with the License (UM-624) variable in the *modelsim.ini* file. Note that settings made from the command line are additive to options set in the License variable.

`-load_elab <filename>`
Loads an elaboration file that was created with **-elab**. Optional. See "Simulating with an elaboration file" (UM-136) for more information.

`-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).

`-nocompress`
Causes VSIM to create uncompressed checkpoint files. Optional. This option may also be specified with the CheckpointCompressMode (UM-622) 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. You can suppress X propagation on individual instances using the **tcheck_set** command (CR-267).

`+no_tchk_msg`
Disables error messages generated when timing checks are violated. Optional. For Verilog, it disables messages issued by timing check system tasks. For VITAL, it overrides the MsgOn arguments and generics.

Notifier registers are still toggled and may result in the propagation of Xs for timing check violations. You can disable individual messages using the **tcheck_set** command (CR-267).

`+notimingchecks`
Disables Verilog and VITAL timing checks for faster simulation. Optional. By default, Verilog timing check system tasks ($setup, $hold,...) in specify blocks are enabled. For VITAL, the timing check default is controlled by the ASIC or FPGA vendor, but most default to enabled. You can disable individual checks using the **tcheck_set** command (CR-267).

`+pulse_int_e/<percent>`
Controls how pulses are propagated through interconnect delays, where <percent> is a number between 0 and 100 that specifies the error limit as a percentage of the interconnect delay. Optional. Used in conjunction with **+multisource_int_delays** (see above). This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog.

A pulse greater than or equal to the error limit propagates to the output in transport mode (transport mode allows multiple pending transitions on an output). A pulse less than the error limit and greater than or equal to the rejection limit (see **+pulse_int_r/<percent>** below) propagates to the output as an X. If the rejection limit is not specified, then it defaults to the error limit. For example, consider an interconnect delay of 10 along with a **+pulse_int_e/80** option. The error limit is 80% of 10 and the rejection limit defaults to 80% of 10. This results in the propagation of pulses greater than or equal to 8, while all other pulses are filtered.

`+pulse_int_r/<percent>`
Controls how pulses are propagated through interconnect delays, where <percent> is a number between 0 and 100 that specifies the rejection limit as a percentage of the interconnect delay. Optional. This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog.

A pulse less than the rejection limit is filtered. If the error limit is not specified by **+pulse_int_e** then it defaults to the rejection limit.

`-quiet`
Disable 'Loading' messages during batch-mode simulation. Optional.

`-restore <filename>`
Specifies that **vsim** is to restore a simulation saved with the **checkpoint** command (CR-99). Optional.

You must restore vsim under the same environment in which you did the checkpoint. This means not only the same type of machine and OS and at least the same memory size, but also the same vsim environment such as GUI vs. command line mode.

`-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. See "Specifying SDF files for simulation" (UM-544).

`-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. See *Chapter 4 - VHDL simulation* for an additional discussion of SDF.

`+sdf_verbose`
Turns on the verbose mode during SDF annotation. The Main window provides detailed warnings and summaries of the current annotation. Optional.

`-t [<multiplier>]<time_unit>`
Specifies the simulator time resolution. Optional. <time_unit> must be one of the following:

fs, ps, ns, us, ms, sec

The default is 1ns; the optional <multiplier> may be 1, 10 or 100. Note that there is no space between the multiplier and the unit (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 (see "Simulator resolution limit" (UM-117) for further details); in Verilog designs without *any* timescale directives, or in a VHDL or mixed design, the value specified for the Resolution (UM-624) 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 (CR-238) with the **simulator state** option.

`-tag <string>`
Specifies a string tag to append to foreign trace filenames. Optional. Used with the **-trace_foreign <int>** option. Used when running multiple traces in the same directory. See the *ModelSim FLI Reference* for more information.

`-title <title>`
Specifies the title to appear for the ModelSim Main window. Optional. If omitted the current ModelSim version is the window title. Useful when running multiple simultaneous simulations. Text strings with spaces must be in quotes (e.g., "my title").

`-trace_foreign <int>`
Creates two kinds of foreign interface traces: a log of what functions were called, with the value of the arguments, and the results returned; and a set of C-language files to replay what the foreign interface side did.

The purpose of the logfile is to aid the debugging of your FLI/PLI/VPI code. The primary purpose of the replay facility is to send the replay file to MTI support for debugging co-simulation problems, or debugging problems for which it is impractical to send the FLI/PLI/VPI code. See the *ModelSim FLI Reference* for more information.

`+transport_int_delays`
Selects transport mode with pulse control for single-source nets (one interconnect path). Optional. By default interconnect delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through interconnect delays.

This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog. This option works independently from **+multisource_int_delays**.

`-vcdstim [<instance>=]<filename>`
Specifies a VCD file from which to re-simulate the design. Optional. The VCD file must have been created in a previous ModelSim simulation using the **vcd dumpports** command (CR-287). See "Using extended VCD as stimulus" (UM-562) for more information.

`-version`
Returns the version of the simulator as used by the licensing tools, such as "Model Technology ModelSim SE vsim 5.5 Simulator 2000.01 Jan 28 2000".

`-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, Signals, 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 "Examples" (CR-372).

`-wlf <filename>`
Specifies the name of the wave log format (WLF) file to create. The default is *vsim.wlf*. Optional.

`-wlfcompress`
Creates compressed WLF files. Default. Use **-wlfnocompress** to turn off compression.

`-wlfopt`
Optimizes the display of waveforms in the Wave window. Default. Optional. WLF files created prior to ModelSim version 5.8 cannot take advantage of the optimization. This option may also be specified with the WLFOptimize (UM-626) variable in the *modelsim.ini* file.

`-wlfnocompress`
Causes **vsim** to create uncompressed WLF files. Optional. Beginning with version 5.5, 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 (UM-626) variable in the *modelsim.ini* file.

`-wlfnoopt`
Disables optimization of waveform display in the Wave window. Optional. Corresponding *.ini* file entry is WLFOptimize.

`-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 (UM-626) 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 least 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 (UM-626) 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.

Arguments, VHDL

`-absentisempty`
Causes VHDL files opened for read that target non-existent files to be treated as empty, rather than ModelSim issuing fatal error messages. Optional.

`-foreign <attribute>`
Specifies the foreign module to load. Optional. <attribute> is a quoted string consisting of the name of a C function and a path to a shared library. For example,

```
vsim -foreign "c_init for.sl"
```

You can load up to ten foreign modules. Syntax for the attribute is further described in the Introduction chapter of the *ModelSim FLI Reference*.

`-nocollapse`
Disables the optimization of internal port map connections. Optional.

`-nofileshare`
Turns off file descriptor sharing. Optional. By default ModelSim shares a file descriptor for all VHDL files opened for write or append that have identical names.

`-noglitch`
Disables VITAL glitch generation. Optional.

See *Chapter 4 - VHDL simulation* for additional discussion of VITAL.

`+no_glitch_msg`
Disable VITAL glitch error messages. Optional.

`-nopsl`
Instructs ModelSim to ignore any PSL assertions that were compiled with the design. By default **vsim** automatically invokes the PSL assertion engine at runtime if any assertions were compiled with the design.

`-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.

`-vcdread <filename>`
Simulates the VHDL top-level design from the specified VCD file. Optional. This argument is included for backwards compatibility. Resimulations in ModelSim versions 5.5c and newer should use the **-vcdstim** argument. See "Simulating with input values from a VCD file" (UM-562) for more details.

`-vital2.2b`
Selects SDF mapping for VITAL 2.2b (default is VITAL 2000). Optional.

Arguments, Verilog

`+alt_path_delays`
Configures path delays to operate in inertial mode by default. Optional. In inertial mode, a pending output transition is cancelled when a new output transition is scheduled. The result is that an output may have no more than one pending transition at a time, and that pulses narrower than the delay are filtered. The delay is selected based on the transition from the cancelled pending value of the net to the new pending value. The **+alt_path_delays** option modifies the inertial mode such that a delay is based on a transition from the current output value rather than the cancelled pending value of the net. This option has no effect in transport mode (see **+pulse_e/<percent>** and **+pulse_r/<percent>**).

`+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. See "Using delayed inputs for timing checks" (UM-125).

`-extend_tcheck_data_limit <percent>`

`-extend_tcheck_ref_limit <percent>`
Causes a one-time extension of qualifying data or reference limits in an attempt to provide a delay net solution prior to any limit zeroing. A limit qualifies if it bounds a violation region which does not overlap a related violation region.

<percent> is the maximum percent of limit relaxation. See "Extending check limits without zeroing" (UM-124) for an example of how to calculate the percentage.

`-hazards`
Enables event order hazard checking in Verilog modules. Optional. You must also specify this argument when you compile your design with **vlog** (CR-345). See "Hazard detection" (UM-122) for more details.

Important: Enabling **-hazards** implicitly enables the **-compat** argument. As a result, using this argument may affect your simulation results.

`+int_delays`
Optimizes annotation of interconnect delays for designs that have been compiled using **-fast** (see **vlog** command (CR-345)). Optional. This argument causes **vsim** to insert "placeholder" delay elements at optimized cell inputs, resulting in faster backannotation of interconnect delays from an SDF file.

`-L <library_name> ...`
Specifies the library to search for design units instantiated from Verilog. See "Library usage" (UM-111) 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. See "Library usage" (UM-111) for more information. Optional.

`+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. You can suppress X propagation on individual instances using the **tcheck_set** command (CR-267).

`+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_show_cancelled_e`
Filters negative pulses on specify path delays so they don’t show on the output. Default. Use **+show_cancelled_e** to drive a pulse error state.

`+no_tchk_msg`
Disables error messages issued by timing check system tasks when timing check violations occur. Optional. Notifier registers are still toggled and may result in the propagation of Xs for timing check violations. You can disable individual messages using the **tcheck_set** command (CR-267).

`+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.

`-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, see "Preference variables located in INI files" (UM-617). 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.

`+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 (see "ModelSim Verilog system tasks" (UM-149) for more information). 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.[a]

a.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

`vsim -gedge=’"low high"’ -gVCC=4.75 cpu`

Invokes **vsim** on the entity *cpu* and assigns values to the generic parameters *edge* and *VCC*. If working within the ModelSim GUI, you would enter the command as follows:

```
vsim {-gedge="low high"} -gVCC=4.75 cpu
```

`vsim -view test=sim2.wlf`

Instructs 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. For example:

```
vsim -wlf my_design.i01 my_asic structure
vsim -wlf my_design.i02 my_asic structure
```

`vsim -sdfmin /top/u1=myasic.sdf`

Annotates instance */top/u1* using the minimum timing from the SDF file *myasic.sdf*.

Use multiple switches to annotate multiple instances:

```
vsim -sdfmin /top/u1=sdf1 -sdfmin /top/u2=sdf2 top
```

`vsim ’mylib.top(only)’ gatelib.cache_set`

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 -do "set PrefMain(forceQuit) 1; run -all" work.test_counter`

Invokes **vsim** on *test_counter* and instructs the simulator to run until a break event and quit when it encounters a $finish task.

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 5.4".

`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 (CR-357).

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
```

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 (see "Time-based breakpoints" (CR-379)). Conditions can include the following HDL items: VHDL signals, and Verilog nets and registers. Use the **nowhen** command (CR-209) to deactivate **when** commands.

The **when** command uses a **when_condition_expression** to determine whether or not to perform the action. 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 item names, signame'event, or constants. ModelSim evaluates the condition every time any item in the condition changes, hence the restrictions.

With no arguments, **when** will list the currently active when statements and their labels (explicit or implicit).

Note: Virtual signals, functions, regions, types, etc. cannot be used in the **when** command. Neither can simulator state variables other than $now.

Syntax

```
when
  [[-label <label>] [-id <id#>] {<when_condition_expression>} {<command>}]
```

Arguments

`-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 (CR-81). 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 item name, in which case the curly braces can be omitted. The command will be executed when the item changes value. The condition can be an expression with these operators:

Name	Operator
equals	==, =
not equal	!=, /=
greater than	>

Name	Operator
less than	<
greater than or equal	>=
less than or equal	<=
AND	&&, AND
OR	||, OR

The operands may be item 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 items 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 (CR-246) cannot be used with the **when** command. Required if a when expression is specified. The command sequence usually contains a **stop** command (CR-265) 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 (CR-265) within your when statement. DO NOT use an **exit** command (CR-171) or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated. See "Ending the simulation with the stop command" (CR-378) for examples.

Examples

The **when** command below instructs the simulator to display the value of item *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"
}
```

This next example uses the Tcl **set** command to disable arithmetic package warnings at time 0. Note that the time unit (ns in this case) would vary depending on your simulation resolution.

```
when {$now = @1ns } {set NumericStdNoWarnings 1}
run -all
```

The **when** command below is labeled *a* and will cause ModelSim to echo the message "b changed" whenever the value of the item *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** (CR-161) and a **stop** (CR-265) 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}
```

If you encounter a vectored net caused by compiling with **-fast,** use the ’event qualifier to prevent the command from falsely evaluating when unrelated bits of ’a’ change:

```
when {a(3:1) = 3'h7 and a(3:1)'event} {echo "matched at time " $now}
```

The first **when** command below sets up a trigger for the falling edge of RESET. When this happens, a second **when** command is executed which sets up a trigger to occur 200us after the current time.

```
force SIGA 1
when {RESET'falling} {
    when {$now == 200us} {
        noforce SIGA
    }
}
run -all
```

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** (CR-161) and a **stop** (CR-265) 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 = @1750ns} {stop}
```

You can also use:

```
when {errorFlag = '1' OR $now = 2ms} {stop}
```

This example adds 2ms to the simulation time at which the **when** statement is first evaluated, then stops.

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 (CR-81), **disablebp** (CR-153), **enablebp** (CR-163), **nowhen** (CR-209)

where

The **where** command displays information about the system environment. This command is useful for debugging problems where ModelSim cannot find the required libraries or support files.

The command displays two system settings:

`current directory`
: This is the current directory that ModelSim was invoked from, or 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.

Syntax

```
where
```

Arguments

None.

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.

Important: This command should be invoked only after you have stopped the simulation using **quit -sim** or **dataset close sim**.

Syntax

```
wlf2log
  [-bits] [-fullname] [-help] [-inout] [-input] [-internal]
  [-l <instance_path>] [-lower] [-o <outfile>] [-output] [-quiet] <wlffile>
```

Arguments

`-bits`
Forces vector nets to be split into 1-bit wide nets in the log file. Optional.

`-fullname`
Shows the full hierarchical pathname when displaying signal names. Optional.

`-help`
Displays a list of command options with a brief description for each. Optional.

`-inout`
Lists only the inout ports. Optional. This may be combined with the -input, -output, or -internal switches.

`-input`
Lists only the input ports. Optional. This may be combined with the -output, -inout, or -internal switches.

`-internal`
Lists only the internal signals. Optional. This may be combined with the -input, -output, or -inout switches.

`-l <instance_path>`
Lists the signals at or below the specified HDL instance path within the design hierarchy. Optional.

`-lower`
Shows all logged signals in the hierarchy. Optional. When invoked without the -lower switch, only the top-level signals are displayed.

`-o <outfile>`
Directs the output to be written to the file specified by <outfile>. Optional. The default destination for the logfile is standard out.

`-output`
Lists only the output ports. Optional. This may be combined with the -input, -inout, or -internal switches.

`-quiet`
Disables error message reporting. Optional.

`<wlffile>`
Specifies the ModelSim WLF file that you are converting. Required.

Additional information for QuickSim II users

In some cases your original QuickHDL/ModelSim simulation results (in your *vsim.wlf* file) may contain signal values that do not directly correspond to qsim_12state values. The resulting QuickSim II logfile generated by **wlf2log** may contain state values that are surrounded by single quotes, e.g. '0' and '1'. To make this logfile compatible with QuickSim models (that expect qsim_12state) you need to use a QuickSim II function named $convert_wdb().

This function was created to convert logfiles resulting from VHDL simulation that used std_logic and std_ulogic since these data types do not correlate to QuickSim's 12 simulation states. Other VHDL data types such as qsim_state or bit (2 state) do not require conversion as they are directly compatible with qsim_12state QuickSim II Waveform Databases (WDB).

The following procedure can be used to convert a wlf2log-generated logfile into a compatible QuickSim WDB. The procedure below shows how to convert the logfile while loaded into memory in QuickSim II.

1. Load the logfile (the output from **wlf2log**) into a WDB other than "forces". "Forces" is the default WDB, so you need to choose a unique name for the WDB when loading the logfile (for example, "fred").

2. Enter the following at the command prompt from within QuickSim:

   ```
   $convert_wdb("fred",0)
   ```

   The first argument, which is "fred", is the name of the new WDB to be created. The second argument, which is 0, specifies the type of conversion. At this time only one type of conversion is supported. The value 0 specifies to convert std_logic or std_ulogic into qsim_12state.

3. Do a connect_wdb (either through the pulldown menus, the "Connect WDB" palette icon under "Stimulus", or a function invocation). You specify the name of the WDB that you originally loaded the logfile into (in this case, "fred").

At this point you can issue the "run" command and the stimulus in the converted logfile will be applied. Before exiting the simulation you should save the new WDB ("fred") as a WDB or logfile so that it can be loaded again in the future. The new WDB or logfile will contain the correct qsim_12state values eliminating the need to re-use $convert_wdb().

wlf2vcd

The **wlf2vcd** command translates a ModelSim WLF file to a standard VCD file. Complex data types that are unsupported in the VCD standard (records, memories, etc) are not converted.

Important: 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 items (i.e., signals) from the source WLF file and outputs it to stdout. When redirected to a file, the output is called an item_list_file, and it can be read in by **wlfman filter**. The item_list_file is a list of items, 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 item_list_file.
- **wlfman filter** reads in a WLF file and optionally an item_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.

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>] [-eend <time>] [-f <item_list_file>] [-r <item>] [-s
  <symbol>]
  [-t <resolution>] <sourcewlffile> <outwlffile>

wlfman profile
  [-rank] [-top <number>] <wlffile>
```

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 item type next to each item. 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.

`-eend <time>`
Specifies the simulation time at which you want to end reading information from the source WLF file. Optional.

`-f <item_list_file>`
Specifies an item-list-file created by **wlfman items** to include in the output WLF file. Optional.

`-r <item>`
Specifies an item (region) to recursively include in the output. If <item> is a signal, the output would be the same as using **-s**. Optional.

`-s <symbol>`
Specifies an item to include in the output. Optional. By default all items 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.

`<sourcewlffile>`
Specifies the source WLF file from which you want items. Required.

`<outwlffile>`
Specifies the name of the output WLF file. Required. The output WLF file will contain all items specified by **-f <item_list_file>**, **-r <item>**, and **-s <symbol>**. Output WLF files are always written in the latest WLF version regardless of the source WLF file version.

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 item information. Required.

Examples

`wlfman profile -rank top_vh.wlf`
The output from this command would look something like this:

```
#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

Chapter 9 - WLF files (datasets) and virtuals (UM-239)

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. The tool works only on WLF files created by ModelSim versions 5.6 or later. 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. Required for PCs.

`-q`
Hides all messages unless there is an error while repairing the file. Optional.

write cell_report

The **write cell_report** command writes to the Main window transcript or to a file a list of optimized (**-fast**) cell instances in the current design.

Syntax

```
write cell_report
  [-filter <number>] [-infile <filename>] [-nonopt]
  [[-outfile] <filename>]
```

Arguments

`-filter <number>`
Excludes cells with instance counts fewer than <number>. Optional.

`-infile <filename>`
Specifies a previously generated write report file to use as input. Optional. If not specified then the write report command will be run.

`-nonopt`
Reports only non-optimized instances. Optional.

`[-outfile] <filename>`
Writes the report to the specified output file rather than the Main window transcript. Optional.

write format

The **write format** command records the names and display options of the HDL items currently being displayed in the List or Wave window. The file created is primarily a list of **add list** (CR-55), **add wave** (CR-64), and **configure** (CR-129) commands, though a few other commands are included (see "Output" below). This file may be invoked with the **do** command (CR-156) 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 [-window <window_name>] <filename>
```

Arguments

`list | wave`
Specifies that the contents of either the List or the Wave window are to be recorded. Required.

`-window <window_name>`
Specifies the window for which you want contents recorded. Optional. Use when you have more than one instance of the List or Wave window.

`<filename>`
Specifies the name of the output file where the data is to be written. Required.

Examples

`write format list alu_list.do`
Saves the current data in the List window in a file named *alu_list.do*.

`write format wave alu_wave.do`
Saves the current data in the Wave window in a file named *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 (CR-55), **add wave** (CR-64)

write list

The **write list** command records the contents of the most recently opened or specified List window in a list output file. This file contains simulation data for all HDL items displayed in the List window: VHDL signals and variables and Verilog nets and registers.

Syntax

```
write list
  [-events] [-window <wname>] <filename>
```

Arguments

`-events`
Specifies to write print-on-change format. Optional. Default is tabular format.

`-window <wname>`
Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the **view** command (CR-320) to change the default window.

`<filename>`
Specifies the name of the output file where the data is to be written. Required.

Examples

`write list alu.lst`
Saves the current data in the default List window in a file named *alu.lst*.

`write list -win list1 group1.list`
Saves the current data in window 'list1' in a file named *group1.list*.

See also

write tssi (CR-395)

write preferences

The **write preferences** command saves the current GUI preference settings to a Tcl preference file. Settings saved include current window locations and sizes; Wave, Signals, and Variables window column widths; Wave, Signals, and Variables 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** (UM-614) environment variable.

See also

You can modify variables by editing the preference file with the ModelSim **notepad** (CR-207):

```
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. If you have compiled a Verilog design using **-fast** (see "Compiling for faster performance" (UM-127)), the report will also identify cells which have been optimized.

Syntax

```
write report
  [[<filename>] [-l | -s]] | [-tcl]
```

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 Main window transcript.

`-l`
Generates more detailed information about the design. 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

`write report alu.rep`
Saves information about the current design in a file named *alu.rep*.

write transcript

The **write transcript** command writes the contents of the Main window transcript 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" (UM-264) 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 (CR-156)

write tssi

The **write tssi** command records the contents of the default or specified List window in a "TSSI format" file. The file contains simulation data for all HDL items displayed in the List window that can be converted to TSSI format (VHDL signals and Verilog nets). A signal definition file is also generated.

The List window needs to be using symbolic radix in order for **write tssi** to produce useful output.

Syntax

```
write tssi
  [-window <wname>] <filename>
```

Arguments

`-window <wname>`
Specifies an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the **view** command (CR-320) to change the default window.

`<filename>`
Specifies the name of the output file where the data is to be written. Required.

Description

"TSSI format" is documented in the Fluence TDS Software System, Chapter 2 of Volume I, Getting Started, R11.1, dated November 15, 1999. In that document, TSSI format is called Standard Events Format (SEF).

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 item is a port, otherwise it is assumed to be bidirectional (mode INOUT).

Items 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

std_ulogic State Characters	SEF State Characters		
	Input	Output	Bidirectional
1	U	H	1
Z	Z	T	F
W	N	X	?
L	D	L	0
H	U	H	1
-	N	X	?

Bidirectional logic values are not converted because only the resolved value is available. The Fluence (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 from Fluence Technology. ModelSim outputs a vector file, and Fluence's tools determine whether the bidirectional signals are driving or not.

See also

tssi2mti (CR-280)

write wave

The **write wave** command records the contents of the most currently opened or specified Wave window in PostScript format. The output file can then be printed on a PostScript printer.

Syntax

```
write wave
  [-window <wname>] [-width <real_num>] [-height <real_num>]
  [-margin <real_num>] [-start <time>] [-end <time>] [-perpage <time>]
  [-landscape] [-portrait] <filename>
```

Arguments

`-window <wname>`
Specifies an instance of the Wave window that is not the default. Optional. Otherwise, the default Wave window is used. Use the **view** command (CR-320) to change the default window.

`-width <real_num>`
Specifies the paper width in inches. Optional. Default is 8.5.

`-height <real_num>`
Specifies the paper height in inches. Optional. Default is 11.0.

`-margin <real_num>`
Specifies the margin in inches. Optional. Default is 0.5.

`-start <time>`
Specifies the start time (on the waveform timescale) to be written. Optional.

`-end <time>`
Specifies the end time (on the waveform timescale) to be written. Optional.

`-perpage <time>`
Specifies the time width per page of output. Optional.

`-landscape`
Use landscape (horizontal) orientation. Optional. This is the default orientation.

`-portrait`
Use portrait (vertical) orientation. Optional. The default is landscape (horizontal).

`<filename>`
Specifies the name of the PostScript output file. Required.

Examples

`write wave alu.ps`
Saves the current data in the Wave window in a file named *alu.ps*.

`write wave -win wave2 group2.ps`
Saves the current data in window ‘wave2’ in a file named *group2.ps*.

```
write wave -start 600ns -end 800ns -perpage 100ns top.ps
```

Writes 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.

To make the job of creating a PostScript waveform output file easier, use the **File > Print Postscript** menu selection in the Wave window. See "Printing and saving waveforms" (UM-363) for more information.

Licensing Agreement

IMPORTANT - USE OF THIS SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS.
CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE SOFTWARE.

This license is a legal "Agreement" concerning the use of Software between you, the end user, either individually or 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 or authorized distributors (collectively "Mentor Graphics"). USE OF SOFTWARE INDICATES YOUR COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. 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.

END-USER LICENSE AGREEMENT

1. **GRANT OF LICENSE.** The software programs you are installing, downloading, or have acquired with this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to 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; and (c) on the computer hardware or at the site for which an applicable license fee is paid, or as 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 service plan purchased, apply to the following and are subject to change: (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 communicated and technically implemented through the use of authorization codes or similar devices); (c) support services provided, including eligibility to receive telephone support, updates, modifications and revisions. Current standard policies and programs are available upon request.

2. **ESD SOFTWARE.** If you purchased a license to use embedded software development ("ESD") Software, 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 or incorporate copies of Mentor Graphics' real-time operating systems or other ESD Software, except those explicitly granted in this section, into your products without first signing a separate agreement with Mentor Graphics for such purpose.

3. **BETA CODE.** Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to 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 conceives 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 subsection shall survive 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 contractors, excluding Mentor Graphics' competitors, whose job performance requires access. 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 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 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.

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.

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 USEOF SOFTWARE AS DESCRIBED IN SECTION 7.

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 9 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 automatically terminate if you fail to comply with any term or condition of this Agreement or if you fail to pay for the license when due and such failure to pay continues for a period of 30 days after written notice from Mentor Graphics. 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 in this Agreement.

14. **AUDIT RIGHTS.** With reasonable prior notice, Mentor Graphics shall have the right to audit during your normal business hours all records and accounts as may contain information regarding your compliance with the terms of this Agreement. Mentor Graphics shall keep in confidence all information gained as a result of any audit. Mentor Graphics shall only use or disclose such information as necessary to enforce its rights under this Agreement.

15. **CONTROLLING LAW AND JURISDICTION.** THIS AGREEMENT SHALL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF OREGON, USA, IF

YOU ARE LOCATED IN NORTH OR SOUTH AMERICA, AND THE LAWS OF IRELAND IF YOU ARE LOCATED OUTSIDE OF NORTH AND SOUTH AMERICA. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of Dublin, Ireland when the laws of Ireland apply, or Wilsonville, Oregon when the laws of Oregon apply. This section shall not restrict Mentor Graphics' right to bring an action against you in the jurisdiction where your place of business is located.

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. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions, except valid license agreements related to the subject matter of this Agreement (which are physically signed by you and an authorized agent of Mentor Graphics) either referenced in the purchase order or otherwise governing this subject matter. 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. The prevailing party in any legal action regarding the subject matter of this Agreement shall be entitled to recover, in addition to other relief, reasonable attorneys' fees and expenses.

Rev. 020826, Part Number 214231

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Index

CR = Command Reference, UM = User's Manual

Symbols

Numerics

A

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

B

C

D

E

F

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

G

H

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

I

K

L

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

N

O

P

Q

R

S

T

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

U

V

A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

W

X

Y

Z