ModelSim® Reference Manual

Software Version 10.1c

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

Table of Contents

List of Examples

List of Figures

List of Tables

Chapter 1
Syntax and Conventions

Documentation Conventions

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

Table 1-1. Conventions for Command Syntax

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

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

   add wave {vector1[4:0]}

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

Note

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

File and Directory Pathnames

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

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

or

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

Design Object Names

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

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

Object Name Syntax

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

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

where

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

(.) nor (/) can be used when referring to the contents of a SystemVerilog package or class.

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

Tcl Syntax and Specification of Array Bits and Slices

Because ModelSim is a Tcl-based tool, you must surround objects and signals with curly braces ({}) when specifying array bits or slices with parentheses (()), spaces, or square brackets ([]). For example:

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

Further Details

Because ModelSim is basd on Tcl, its commands follow Tcl syntax. One problem you may encounter with ModelSim commands is the use of square brackets ([]), parentheses (()), or spaces when specifying array bits and slices. As shown on the previous page, square brackets are used to specify slices of arrays (for example, *data[3:0]*). However, in Tcl, square brackets signify command substitution. Consider the following example:

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

ModelSim evaluates the **find** command first and then sets variable *aluinputs* to the result of the find command. Obviously you 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)}
```

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

SystemVerilog Scope Resolution Operator

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

Example 1-1. SystemVerilog Scope Resolution Operator Example

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

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

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

The following examine examples access data from the class *packet*.

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

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

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

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

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

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

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

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

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

Specifying Names

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

- **Simple name** — does not contain any hierarchy. It is simply the name of an object (e.g., *clk* or *data[3:0]*) in the current context.
- **Relative name** — does not start with a path separator and may or may not include a dataset name or a hierarchical path (e.g., *u1/data* or *view:clk*). A relative name is relative to the current context in the current or specified dataset.
- **Fully-rooted name** — starts with a path separator and includes a hierarchical path to an object (e.g., */top/u1/clk*).There is a special case of a fully-rooted name where the top-level design unit name can be unspecified (e.g., */u1/clk*). In this case, the first top-level instance in the design is assumed.
- **Absolute name** — is an exactly specified hierarchical name containing a dataset name and a fully rooted name (e.g., *sim:/top/u1/clk*).

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

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

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

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

```
package p;
```

```
    int I;
    function automatic int factorial(int n);
       if(n==0)
     return 1;
       else
     return  n * factorial(n - 1);
    endfunction : factorial
endpackage : p
module top;
    initial begin
       p::I=p::factorial(3);

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

When you issue the command:

examine n

the transcript returns:

```
0
```

However, when you issue the command:

stack up;examine n

the transcript returns:

```
1
```

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

Table 1-2. Examples of Object Names

Object Name	Description
clk	specifies the object *clk* in the current context
/top/clk	specifies the object *clk* in the top-level design unit.
/top/block1/u2/clk	specifies the object *clk*, two levels down from the top-level design unit
block1/u2/clk	specifies the object *clk*, two levels down from the current context
array_sig[4]	specifies an index of an array object
{array_sig(1 to 10)}	specifies a slice of an array object in VHDL; see Tcl Syntax and Specification of Array Bits and Slices for more information

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

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

Environment Variables and Pathnames

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

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

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

Name Case Sensitivity

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

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

Extended Identifiers

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

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

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

Wildcard Characters

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

- add dataflow
- add list

- add memory
- add watch
- add wave
- find
- log

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

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

You can alter these exclusions with the WildcardFilter preference variable. Refer to the section "Using the WildcardFilter Preference Variable" for more information.

Table 1-3 identifies these supported wildcard characters.

Table 1-3. Wildcard Characters in HDL Object Names

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

Note

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

Using the WildcardFilter Preference Variable

The WildcardFilter preference variable controls which object types are excluded when performing wildcard matches with simulator commands. The WildcardFilter preference variable is a Tcl List and can be modified using Tcl commands. You can add both individual

(Figure 1-4) and group variables (Figure 1-5) to the current variable list, and you can remove individual variables from the current list.

Note

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

Procedure

Determining the Current WildcardFilter Variable Settings

Enter one of the following commands:

```
set WildcardFilter
```

or

```
echo $WildcardFilter
```

which returns the list of currently set variables.

Changing the WildcardFilter Settings from the Command Line

Refer to the list of WildcardFilter arguments in Table 1-4 and Figure 1-5 to determine what you want to include in the wildcard matches.

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

  ```
  set WildcardFilter “<arg1 arg2 ...>”
  ```

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

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

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

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

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

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

Changing the WildcardFilter Settings back to the Default

Enter the following command:

```
set WildcardFilter default
```

Changing the WildcardFilter settings from the GUI

1. Choose **Tools** > **Wildcard Filter** from the main menu.

2. Select the individual Filters you want to exclude from wildcard searches (Table 1-4 describes each option), or select Composite Filters to activate related filters (Table 1-5 describes each composite option).
3. Click OK.

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

WildcardFilter Argument Descriptions

Table 1-4 provides a list of the WildcardFilter arguments.

Table 1-4. WildcardFilter Arguments

Argument	Description
Alias	VHDL Alias
Assertion	Concurrent SystemVerilog or PSL assertion
CellInternal	Signals in cells, where a cell is defined as 1) a module within a ‘celldefine 2) a Verilog module found with a library search (using either vlog -v or vlog -y) and compiled with vlog +libcell or 3) a module containing a specify block
Class	Verilog class declaration
ClassReference	SystemVerilog class reference
Compare	Waveform comparison signal
Constant	VHDL constant
Cover	SystemVerilog or PSL cover statements
Covergroup	SystemVerilog or PSL covergroup
Coverpoint	Verilog coverpoint
Cross	Verilog cross
Endpoint	SystemVerilog assertion objects created for sequences on which the method “ended/triggered” is used. PSL assertion objects created for sequences for which the built in function “ended()” is used.
Generic	VHDL generic
ImmediateAssert	VHDL immediate assertions
Integer	VHDL integer
Memory	Verilog memories
NamedEvent	Verilog named event
Net	Verilog net

Table 1-4. WildcardFilter Arguments (cont.)

Argument	Description
Parameter	Verilog parameter
Real	Verilog real registers
Reg	Verilog register
ScVariable	SystemC variable
Signal	VHDL signal
SpecParam	Verilog specparam
Time	Verilog time registers
Transaction	Transaction stream and stream arrays
Variable	VHDL shared variables in packages and design units.
VirtualExpr	Virtual expression
VirtualSignal	Virtual signal

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

Table 1-5. WildcardFilter Argument Groups

Group Argument	Specific arguments included
AllVHDL	Signal, Variable, Constant, Generic, Alias
AllVerilogVars	Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, ClassReference
AllVerilog	Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, Class, Cross, Covergroup, Coverpoint, ClassReference
VirtualSignals	VirtualSignal, VirtualExpr
SystemC	ScVariable
AllHDLSignals	Signal, Variable, Constant, Generic, Alias, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, VirtualSignal, VirtualExpr, ClassReference
AllVariables	Variable, Constant, Generic, Alias, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, ClassReference
AllHDLSignalsVars	Signal, Variable, Constant, Generic, Alias, Net, Parameter, Reg, Integer, Time, Real, SpecParam, Memory, NamedEvent, VirtualSignal, VirtualExpr, ClassReference

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

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

Simulator Variables

You can reference ModelSim variables in a simulator command by preceding the name of the variable with the dollar sign ($) character. ModelSim uses global variables for simulator state variables, simulator control variables, simulator preference variables, and user-defined variables. Refer to modelsim.ini Variables in the User's Manual for more information on variables.

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

Simulation Time Units

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

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

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

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

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

Argument Files

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

```
vlog   vcom   vencrypt   vmake   vsim
```

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

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

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

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

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

- Unquoted — The following are notes on what occurs when some information is not quoted:
  - Tcl backslash substitution — Any unquoted backslash (\ ) will be treated as an escape character.

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

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

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

- Newline Character — You can specify arguments on separate lines in the argument file with the line continuation character (\ ). You must use a space before the backslash.
- Comments — Comments within the argument files follow these rules:
  - All text in a line beginning with // to its end is treated as a comment.
  - All text bracketed by /* … */ is treated as a comment.
  - All text in a line beginning with # to its end is treated as a comment.

Command Shortcuts

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

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

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

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

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

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

Command History Shortcuts

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

Table 1-6. Keyboard Shortcuts for Command History

Shortcut	Description
!!	repeats the last command
!n	repeats command number n; n is the VSIM prompt number (e.g., for this prompt: VSIM 12>, n =12)
!<string>	shows a list of executed commands that start with <string>. Use the up and down arrows to choose from the list.
!abc	repeats the most recent command starting with "abc"
^xyz^ab^	replaces "xyz" in the last command with "ab"
up 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

Table 1-6. Keyboard Shortcuts for Command History (cont.)

Shortcut	Description
his or history	shows the last few commands (up to 50 are kept)

Numbering Conventions

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

VHDL Numbering Conventions

There are two types of VHDL number styles:

VHDL Style 1

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

Table 1-7. VHDL Number Conventions: Style 1

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

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

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

VHDL Style 2

```
base "value"
```

Table 1-8. VHDL Number Conventions: Style 2

Element	Description
base	specifies the base; binary: B, octal: O, hex: X; required

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

Element	Description
"value"	specifies digits in the appropriate base with optional underscore separators; default is decimal; required

For example:

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

Searching for VHDL Arrays in the Wave and List Windows

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

Verilog Numbering Conventions

Verilog numbers are expressed in the style:

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

Table 1-9. Verilog Number Conventions

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

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

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

GUI_expression_format

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

configure, examine, searchlog, virtual function, virtual signal.

Expression Typing

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

Scalar Types

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

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

Array Types

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

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

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

Expression Syntax

GUI expressions generally follow C-language syntax, with both VHDL-specific and Verilog-specific conventions supported. These expressions are not parsed by the Tcl parser, and so do

not support general Tcl; parentheses should be used rather than braces. Procedure calls are not supported.

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

Tcl Macros

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

```
$<name>
```

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

Constants

Table 1-10. Constants Supported for GUI Expresssions

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

Array Constants, Expressed in Any of the Following Formats

Table 1-11. Array Constants Supported for GUI Expresssions

Type	Values
VHDL # notation	<int>#<alphanum>[#] Example: 16#abc123#
VHDL bitstring	"(U|X|0|1|Z|W|L|H|-)*" Example: "11010X11"
Verilog notation	[-][<int>]'(b|B|o|O|d|D|h|H) <alphanum> (where <alphanum> includes 0-9, a-f, A-F, and '-') Example: 12'hc91 (This is the preferred notation because it removes the ambiguity about the number of bits.)
Based notation	0x…, 0X…, 0o…, 0O…, 0b…, OB… ModelSim automatically zero fills unspecified upper bits.

Variables

Table 1-12. Variables Supported for GUI Expresssions

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

Table 1-12. Variables Supported for GUI Expresssions (cont.)

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

Array variables

Table 1-13. Array Variables Supported for GUI Expresssions

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

Signal attributes

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

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

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

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

Operators

Table 1-14. Operators Supported for GUI Expresssions

Operator	Description	Kind
+	arithmetic add	arithmetic
/	arithmetic divide	arithmetic
mod/MOD	arithmetic modulus	arithmetic
*	arithmetic multiply	arithmetic
rem/REM	arithmetic remainder	arithmetic
-	arithmetic subtract	arithmetic
&	concat	arithmetic
<name>'delayed(<time>)	delayed signal (<time>)	attributes
<name>'falling	Falling edge	attributes
<name>'rising	Rising edge	attributes
<name>'event	Value change	attributes
<name>'hasX	Value has an X	attributes
and, AND	bitwise and	bitwise logical
nand, NAND	bitwise nand	bitwise logical
nor, NOR	bitwise nor	bitwise logical
or, OR	bitwise or	bitwise logical
xnor, XNOR	bitwise xnor	bitwise logical
xor, XOR	bitwise xor	bitwise logical
rol, ROL	rotate left	bitwise logical
ror, ROR	rotate right	bitwise logical
sla, SLA	shift left arithmetic	bitwise logical
sll, SLL	shift left logical	bitwise logical
sra, SRA	shift right arithmetic	bitwise logical
srl, SRL	shift right logical	bitwise logical
not, NOT, ~	unary bitwise inversion	bitwise logical
&&	boolean and	boolean
!	boolean not	boolean
||	boolean or	boolean

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

Operator	Description	Kind
==	equal	boolean
===	exact equal[1]	boolean
!==	exact not equal[1]	boolean
>	greater than	boolean
>=	greater than or equal	boolean
<	less than	boolean
<=	less than or equal	boolean
!=, /=	not equal	boolean
&<vector_expr>	AND reduction	reduction
|<vector_expr>	OR reduction	reduction
^<vector_expr>	XOR reduction	reduction

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

Table 1-15. Precedence of GUI Expression Operators

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

Table 1-15. Precedence of GUI Expression Operators (cont.)

Operator	Kind
||	boolean

Note

Arithmetic operators use the std_logic_arith package.

Casting

Table 1-16. Casting Conversions Supported for GUI Expresssions

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

Examples of Expression Syntax

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Signal and Subelement Naming Conventions

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

Examples in Verilog and VHDL syntax:

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

Grouping and Precedence

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

Concatenation of Signals or Subelements

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

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

Concatenation Syntax for VHDL

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

Concatenation Syntax for Verilog

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

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

Concatenation Directives

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

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

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

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

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

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

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

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

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

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

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

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

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

Examples of Concatenation

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

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

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

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

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

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

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

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

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

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

Record Field Members

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

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

Examples:

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

Searching for Binary Signal Values in the GUI

When you use the GUI to search for signal values displayed in 4-state binary radix, you should be aware of how ModelSim maps between binary radix and std_logic. The issue arises because there is no “un-initialized” value in binary, while there is in std_logic. So, ModelSim relies on mapping tables to determine whether a match occurs between the displayed binary signal value and the underlying std_logic value.

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

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

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

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

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

Table 1-18. Verilog Logic Values Used in GUI Search

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

Chapter 2
Commands

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

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

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

Table 2-1. Supported Commands

Command name	Action
abort	halts the execution of a macro file interrupted by a breakpoint or error
add dataflow	adds the specified object(s) to the Dataflow window
add list	lists VHDL signals and variables, and Verilog nets and registers, and their values in the List window
add log	also known as the **log** command; see log
add memory	opens the specified memory in the MDI frame of the Main window
add message	used within a macro or script and specifies a user defined runtime message that is sent to the transcript and *.wlf* files.
add watch	adds signals or variables to the Watch window
add wave	adds VHDL signals and variables, and Verilog nets and registers to the Wave window
add_cmdhelp	adds an entry to the command-line help; use the help command to display the help text
alias	creates a new Tcl procedure that evaluates the specified commands
archive load	allows you to load an archived debug database (.dbg) file that was previously created with the archive write command

Table 2-1. Supported Commands (cont.)

Command name	Action
archive write	allows you to create a debug archive file, with the file extension .dgb, that contains one or more WLF files, debug information captured from the design library, an optional connectivity debug database file, and optional HDL source files.
batch_mode	returns a 1 if ModelSim is operating in batch mode, otherwise returns a 0
bd	deletes a breakpoint
bookmark add wave	adds a bookmark to the specified Wave window
bookmark delete wave	deletes bookmarks from the specified Wave window
bookmark goto wave	zooms and scrolls a Wave window using the specified bookmark
bookmark list wave	displays a list of available bookmarks
bp	sets a breakpoint
call	calls SystemVerilog static, package, and class functions
change	modifies the value of a VHDL variable or Verilog register variable
classinfo	displays information about class types and instances
configure	invokes the List or Wave widget configure command for the current default List or Wave window
dataset alias	assigns an additional name to a dataset
dataset clear	clears the current simulation WLF file
dataset close	closes a dataset
dataset config	configures WLF file settings after dataset is open
dataset current	opens the specified dataset and sets the GUI context to the last selected context of the specified dataset.
dataset info	reports information about the specified dataset
dataset list	lists the open dataset(s)
dataset open	opens a dataset and references it by a logical name
dataset rename	changes the logical name of an opened dataset
dataset restart	unloads specified or current dataset
dataset save	saves data from the current WLF file to a specified file
dataset snapshot	saves data from the current WLF file at a specified interval
delete	removes objects from either the List or Wave window

Table 2-1. Supported Commands (cont.)

Command name	Action
describe	displays information about the specified HDL object
disablebp	turns off breakpoints and when commands
do	executes commands contained in a macro file
drivers	displays in the Main window the current value and scheduled future values for all the drivers of a specified VHDL signal or Verilog net
dumplog64	dumps the contents of the *vsim.wlf* file in a readable format
echo	displays a specified message in the Main window
edit	invokes the editor specified by the EDITOR environment variable
enablebp	turns on breakpoints and when commands turned off by the disablebp command
encoding	translates between Unicode Tcl strings and a named encoding
environment	displays or changes the current dataset and region environment
examine	examines one or more objects, and displays current values (or the values at a specified previous time) in the Main window
exit	exits the simulator and the ModelSim application
find	displays the full pathnames of all objects in the design whose names match the name specification you provide
find infiles	searches the specified files and prints to the Transcript window those lines from the files that match the specified pattern.
find insource	searches all source files related to the current design and prints to the Transcript window those lines from the files that match the specified pattern.
force	applies stimulus to VHDL signals and Verilog nets
formatTime	global format control for all time values displayed in the GUI
help	displays in the Main window a brief description and syntax for the specified command
history	lists the commands executed during the current session
layout	allows you to perform operations on GUI layouts
log	creates a wave log format (WLF) file containing simulation data for all objects whose names match the provided specifications
lshift	takes a Tcl list as an argument and shifts it in-place one place to the left, eliminating the left-most element

Table 2-1. Supported Commands (cont.)

Command name	Action
lsublist	returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern
mem compare	compares the selected memory to a reference memory or file
mem display	displays the memory contents of a selected instance to the screen
mem list	displays a flattened list of all memory instances in the current or specified context after a design has been elaborated
mem load	updates the simulation memory contents of a specified instance
mem save	saves the contents of a memory instance to a file in any of the supported formats: Verilog binary, Verilog hex, and MTI memory pattern data
mem search	finds and prints to the screen the first occurring match of a specified memory pattern in the specified memory instance
messages clearfilter	removes any filter you have set in the Message Viewer
messages setfilter	performs the same action as the Message Viewer Filter Dialog Box, lets you determine which messages are shown in the Message Viewer
messages write	prints the contents of the Message Viewer window to a specified text file.
modelsim	starts the ModelSim GUI without prompting you to load a design; valid only for Windows platforms
noforce	removes the effect of any active force commands on the selected object
nolog	suspends writing of data to the WLF file for the specified signals
notepad	opens a simple text editor
noview	closes a window or set of windows in the ModelSim GUI
nowhen	deactivates selected when commands
onbreak	specifies command(s) to be executed when running a macro that encounters a breakpoint in the source code; in effect only during a run command
onElabError	specifies one or more commands to be executed when an error is encountered during elaboration; in effect only during a vsim command
onerror	specifies one or more commands to be executed when a Tcl command in a dofile encounters an error; not dependent on a run command

Table 2-1. Supported Commands (cont.)

Command name	Action
onfinish	controls simulator behavior when encountering $finish or sc_stop() in the design code
pause	interrupts the execution of a macro
precision	determines how real numbers display in the GUI
printenv	echoes to the Main window the current names and values of all environment variables
process report	creates textual report of all processes displayed in the Process window
project	performs common operations on new projects
pwd	displays the current directory path in the Main window
quietly	turns off transcript echoing for the specified command
quit	exits the simulator
radix	specifies the default radix to be used
radix define	creates or modifies a user-defined radix
radix delete	removes the radix definition from the named radix
radix list	returns the complete definition of a radix
radix names	returns a list of currently defined radix names
radix signal	sets or inspects radix values for the specified signal in the Objects, Locals, and Wave windows
report	displays the value of all simulator control variables, or the value of any simulator state variables relevant to the current simulation
restart	reloads the current dataset if the current dataset is not the active simulation ("sim") and resets the simulation time to zero, in effect acting just like a restart of a simulation
resume	continues execution of a macro file after a pause command or a breakpoint
run	advances the simulation by the specified number of timesteps
runStatus	returns the current state of your simulation after issuing a run or step command
searchlog	searches one or more of the currently open logfiles for a specified condition
see	displays the specified number of source file lines around the current execution line

Table 2-1. Supported Commands (cont.)

Command name	Action
setenv	sets an environment variable
shift	shifts macro parameter values down one place
show	lists objects and subregions visible from the current environment
simstats	reports performance-related statistics about active simulations
stack down	moves down the call stack
stack frame	selects the specified call frame
stack level	reports the current call frame number
stack tb	is an alias for the tb command
stack up	moves up the call stack
status	lists all currently interrupted macros
step	steps to the next HDL statement
stop	stops simulation in batch files; used with the when command
suppress	prevents the specified message(s) from displaying
tb	displays a stack trace for the current process in the Transcript window
Time	performs various numerical comparisons, operations, and conversions on simulation time values
transcript	controls echoing of commands executed in a macro file; also works at top level in batch mode
transcript file	sets or queries the pathname for the transcript file
transcript path	returns the full pathname to the current transcript file
transcript sizelimit	sets or queries the current value for the transcript fileSizeLimit value, saves file when limit is reached, and continues simulation
tssi2mti	converts a vector file in Technology Standard Events Format (TSSI) into a sequence of force and run commands
ui_VVMode	specifies behavior when encountering user interface registration calls used by verification packages, such as AVM or OVM
unsetenv	deletes an environment variable
vcd add	adds the specified objects to the VCD file
vcd checkpoint	dumps the current values of all VCD variables to the VCD file
vcd comment	inserts the specified comment in the VCD file
vcd dumpports	creates a VCD file that captures port driver data

Table 2-1. Supported Commands (cont.)

Command name	Action
vcd dumpportsall	creates a checkpoint in the VCD file that shows the current values of all selected ports
vcd dumpportsflush	flushes the VCD buffer to the VCD file
vcd dumpportslimit	specifies the maximum size of the VCD file
vcd dumpportsoff	turns off VCD dumping and records all dumped port values as x
vcd dumpportson	turns on VCD dumping and records the current values of all selected ports
vcd file	specifies the filename and state mapping for the VCD file created by a vcd add command
vcd files	specifies filenames and state mapping for the VCD files created by the vcd add command; supports multiple VCD files
vcd flush	flushes the contents of the VCD file buffer to the VCD file
vcd limit	specifies the maximum size of the VCD file
vcd off	turns off VCD dumping and records all VCD variable values as x
vcd on	turns on VCD dumping and records the current values of all VCD variables
vcd2wlf	translates VCD files into WLF files
vcom	compiles VHDL design units
vdel	deletes a design unit from a specified library
vdir	lists the contents of a design library
vencrypt	encrypts Verilog code contained within encryption envelopes
verror	prints a detailed description of a message number
vgencomp	writes the equivalent VHDL component declaration for a Verilog module to standard output
vhencrypt	encrypts VHDL code contained within encryption envelopes
view	opens a ModelSim window and brings it to the front of the display
virtual count	counts the number of currently defined virtuals that were not read in using a macro file
virtual define	prints the definition of a virtual signal or function in the form of a command that can be used to re-create the object
virtual delete	removes the matching virtuals
virtual describe	prints a complete description of the data type of one or more virtual signals

Table 2-1. Supported Commands (cont.)

Command name	Action
virtual expand	produces a list of all the non-virtual objects contained in the virtual signal(s)
virtual function	creates a new signal that consists of logical operations on existing signals and simulation time
virtual hide	causes the specified real or virtual signals to not be displayed in the Objects window
virtual log	causes the sim-mode dependent signals of the specified virtual signals to be logged by the simulator
virtual nohide	redisplays a virtual previously hidden with **virtual hide**
virtual nolog	stops the logging of the specified virtual signals
virtual region	creates a new user-defined design hierarchy region
virtual save	saves the definitions of virtuals to a file
virtual show	lists the full path names of all the virtuals explicitly defined
virtual signal	creates a new signal that consists of concatenations of signals and subelements
virtual type	creates a new enumerated type
vlib	creates a design library
vlog	compiles Verilog design units and SystemVerilog extensions
vmake	creates a makefile that can be used to reconstruct the specified library
vmap	defines a mapping between a logical library name and a directory
vsim	loads a new design into the simulator
vsim<info>	returns information about the current vsim executable
vsim_break	stop the current simulation before completion
vsource	specifies an alternative file to use for the current source file
wave	commands for manipulating cursors, for zooming, and for adjusting the wave display view in the Wave window
when	instructs ModelSim to perform actions when the specified conditions are met
where	displays information about the system environment
wlf2log	translates a ModelSim WLF file to a QuickSim II logfile
wlf2vcd	translates a ModelSim WLF file to a VCD file

Table 2-1. Supported Commands (cont.)

Command name	Action
wlfman	outputs information about or a new WLF file from an existing WLF file
wlfrecover	attempts to repair an incomplete WLF file
write format	records the names and display options in a file of the objects currently being displayed in the List or Wave window
write list	records the contents of the List window in a list output file
write preferences	saves the current GUI preference settings to a Tcl preference file
write report	prints a summary of the design being simulated
write timing	prints timing information about the specified instance
write transcript	writes the contents of the Main window transcript to the specified file
write tssi	records the contents of the List window in a “TSSI format” file
write wave	records the contents of the Wave window in PostScript format

abort

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

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

Syntax

abort [<n> | all]

Arguments

- <n>

  (optional) The number of nested macro levels to abort. Specified as an integer greater than 0, where the default value is 1.

- all

  (optional) Instructs the tool to abort all levels of nested macros.

Related Topics

- onbreak
- onElabError
- onerror

add dataflow

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

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

Syntax

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

Arguments

- **<object>** ...

  (required unless specifying **-connect**) Specifies a process, signal, net, or register to add to the Dataflow window. Wildcards are allowed. Multiple objects are specified as a space separated list, Refer to the section "Wildcard Characters" for wildcard usage as it pertains to the add commands. Must be specified as the first argument to the **add dataflow** command.

- -connect <source_net> <destination_net>

  (optional) Computes and displays in the Dataflow window all paths between two nets.

  <source_net> — The net that originates the path search.

  <destination_net> — The net that terminates the path search.

- -in

  (optional) Specifies to add ports of mode IN.

- -inout

  (optional) Specifies to add ports of mode INOUT.

- -out

  (optional) Specifies to add ports of mode OUT.

- -ports

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

- -internal

  (optional) Specifies to add internal (non-port) objects.

- -nofilter

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

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

- -recursive

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

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

Examples

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

  ```
  add dataflow -r /*
  ```

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

  ```
  add dataflow *
  ```

Related Topics

- Automatically Tracing All Paths Between Two Nets
- Dataflow Window
- Using the WildcardFilter Preference Variable

add list

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

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

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

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

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

Syntax

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

Arguments

- **<object> ...**

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

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

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

  ```
  add list myproc/int1
  ```

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

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

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

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

  For example:

  ```
  add list {mybus {a b y}}
  ```

- -allowconstants

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

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

- -depth <level>

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

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

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

- -in

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

- -inout

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

- -out

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

- -ports

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

- -internal

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

- -label <name>

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

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

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

- -nodelta

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

- -<radix_type>

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

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

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

- -radix <type>

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

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

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

- -radixenumnumeric

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

- -radixenumsymbolic

  (optional) Reverses the action of **-radixenumnumeric** and sets the global setting of the default radix (the DefaultRadix variable in the *modelsim.ini* file) to symbolic.

- -recursive

  (optional) *For use with wildcard searches*. Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region. You can use the **-depth** argument to specify how far down the hierarchy to descend. You can use "-r" as an alias to this switch.

- -trigger | -notrigger

  (optional) Specifies whether objects should be updated in the List window when the objects change value.

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

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

- -width <integer>

  (optional) Formats the column width.

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

Examples

- List all objects in the design.

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

- List all objects in the region.

```
add list *
```

- List all input ports in the region.

```
add list -in *
```

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

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

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

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

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

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

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

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

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

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

Related Topics

- add wave
- log
- Extended Identifiers
- Using the WildcardFilter Preference Variable

add memory

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

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

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

Syntax

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

Arguments

- -addressradix {decimal | hex}

  (optional) Specifies the address radix for the memory display.

  decimal — (default) Sets the radix to decimal. You can abbreviate this argument to "d".

  hex — Sets the radix to hexadecimal. You can abbreviate this argument to "h".

- -dataradix <type>

  (optional) Specifies the data radix for the memory display. If you do not specify this switch, the command uses the global default radix.

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

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

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

- -radixenumnumeric

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

- -radixenumsymbolic

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

- -wordsperline <num>

  (optional) Specifies how many words are displayed on each line in the memory window. By default, the information displayed will wrap depending on the width of the window.

  num — Any positive integer

- **<object_name> ...**

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

  Wildcard characters are allowed.

Note

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

Related Topics

- Memory List Window
- Using the WildcardFilter Preference Variable

add message

This command is used within a macro or script and specifies a user defined runtime message that is sent to the transcript and *.wlf* files. Messages are displayed in the Message Viewer window in the GUI. Refer to "Message Viewer Window" for information.

Syntax

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

Arguments

- <message_body>

  (required) User specified message.

- -category <category>

  (optional) Sets the category for the message in the Message Viewer window where the default is USER. The Message Viewer window Category column recognizes the following keywords:

Table 2-2. Message Viewer Categories

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

- -efftime <time>

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

  <time> — Specified as an integer or decimal number.

- -file <filename>

  (optional) Displays a user specified string in the File Info column of the Message Viewer window.

- -id <id_number>

  (optional) Assigns an identification number to the message.

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

- -inline

  (optional) Causes the message to also be returned to the caller as the return value of the **add message** command.

- -line <linenumber>

  (optional) Displays the user specified number in File Info column of the Message Viewer window.

- -noident

  (optional) Prevents return of the ID number of the message.

- -nolevel

  (optional) Prevents return of the severity level of the message.

- -objects <list>

  (optional) List of related design items shown in the Objects column of the Message Viewer window.

  <list> — A space separated list enclosed in curly braces ({}) or quotation marks (" ").

- -region region

  (optional) Message is displayed in the Region column of the Message Viewer window.

- -severity {error | note | warning}

  (optional) Sets the message severity level.

  error — ModelSim cannot complete the operation.

  note — (default) The message is informational.

  warning — There may be a problem that will affect the accuracy of the results.

Examples

- Create a message numbered 80304.

add message -id 304 <message>

Related Topics

- displaymsgmode modelsim.ini variable
- msgmode modelsim.ini variable
- Message Viewer Window

add watch

This command adds signals and variables to the Watch window in the Main window.

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

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

Syntax

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

Arguments

- **<object_name> ...**

  (required) Specifies the name of the object to be added. Multiple objects are entered as a space separated list. Must be specified as the first argument to the **add watch** command.

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

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

  ```
  add watch myproc/int1
  ```

- -radix <type>

  (optional) Specifies a user-defined radix. If you do not specify this switch, the command uses the global default radix.

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

  You can change the default radix for the current simulation using the **radix** command. You can change the default radix permanently by editing the DefaultRadix variable in the *modelsim.ini* file.

- -radixenumnumeric

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

- -radixenumsymbolic

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

Related Topics

- Watch window
- Using the WildcardFilter Preference Variable
- Wildcard Characters
- DefaultRadix variable

add wave

This command adds the following objects to the Wave window:

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

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

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

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

Syntax

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

Arguments

- -allowconstants

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

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

- -clampanalog {0 | 1}

  (optional) Clamps the display of an analog waveform to the values specified by **-max** and **-min**. Specifying a value of 1 prevents the waveform from extending above the value specified for **-max** or below the value specified for **-min**.

  0 — not clamped

  1 — (default) clamped

- -color <standard_color_name>

  (optional) Specifies the color used to display a waveform.

  <standard_color_name> — You can use either of the following:

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

  -color "light blue"

rgb value — for example:

-color #357f77

- -depth <level>

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

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

- -divider [<divider_name> ...]

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

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

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

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

- -expand <signal_name>

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

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

- -format <type> | -<format>

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

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

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

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

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

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

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

- -height <pixels>

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

  <pixels> — Any positive integer.

- -in

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

- -out

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

- -inout

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

- -ports

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

- -internal

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

- -label <name>

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

  add wave -label c clock

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

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

- -max <real_num>

  (optional) Specifies the maximum Y-axis data value to be displayed for an analog waveform. Used in conjunction with the **-min** switch; the value you specify for **-max** must be greater than the value you specify for **-min**.

  <real_num> — Any integer that is greater than the value specified for **-min**.

- -min <real_num>

  (optional) Specifies the minimum Y-axis data value to be displayed for an analog waveform. Used in conjunction with the **-max** switch; the value you specify for **-min** must be less than the value you specify for **-max**.

  <real_num> — Any integer that is less than the value specified for **-max**.

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

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

Note

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

- -noupdate

  (optional) Prevents the Wave window from updating when a series of add wave commands are executed in series.

- -position <location>

  (optional) Specifies where the command adds the signals.

  <location> — Can be any of the following:

  top — Adds the signals to the beginning of the list of signals.

  bottom | end — Adds the signals to the end of the list of signals.

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

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

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

- -<radix_type>

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

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

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

- -radix <type>

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

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

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

- -radixenumnumeric

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

- -radixenumsymbolic

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

- -recursive

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

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

- -time

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

- **<object_name> ...**

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

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

  ```
  add wave myproc/int1
  ```

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

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

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

Note

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

Examples

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

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

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

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

- Add all wave objects in the region.

  ```
  add wave *
  ```

- Add all wave input ports in the region.

  ```
  add wave -in *
  ```

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

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

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

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

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

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

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

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

- Add an unnamed divider.

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

Related Topics

- add list
- log
- Concatenation Directives
- Extended Identifiers
- Using the WildcardFilter Preference Variable
- Refer to Wave Window for more information on analog formats of waveform signals.

add_cmdhelp

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

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

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

Syntax

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

Arguments

- {<**command_name**>}

  (required) Specifies the command name that will be entered as an argument to the **help** command. Must be enclosed in braces ({ }). The command_name must not interfere with an already existing command_name. Must be specified as the first argument to the **add_cmdhelp** command.

- {<**command_description**>}

  (required) Specifies a description of the command. Must be enclosed in braces ({ }). Must be specified as the second argument to the **add_cmdhelp** command.

- {<**command_arguments**>}

  (required) A space-separated list of arguments for the command. Must be enclosed in braces ({ }). If the command doesn't have any arguments, enter {}. Must be specified as the third argument to the **add_cmdhelp** command.

Examples

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

  add_cmdhelp date {Displays date and time.} {}

  Entering:

  VSIM> help date

  returns:

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

- Add the change date command.

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

  Entering:

  VSIM> help change date

returns:

```
Modify data or time.
Usage: change date -time|-date <arg>
```

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

  add_cmdhelp {change date} {} {}

alias

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

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

Syntax

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

Arguments

- <name>

  (optional) Specifies the new procedure name to be used when invoking the commands.

- "<cmds>"

  (optional) Specifies the command or commands to be evaluated when the alias is invoked. Multiple commands are specified as a semicolon (;) separated list. You must enclose the string in quotes (“”).

Examples

- List all aliases currently defined.

  alias

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

  alias <name>

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

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

archive load

The archive load command allows you to load an archived debug database (.dbg) file that was previously created with the archive write command.

Syntax

archive load <archive_name> [-dbgDir <directory_name>] -wlfFiles <wlf_file_name>

Arguments

- <archive_name>

  Specifies the name of the archived file to be opened for reading.

- -dbgDir <directory_name>

  (optional) Specifies a location to extract files into. Files are extracted on-demand when Modelsim needs them. The current working directory is used if the switch is not specified.

- -wlfFiles <wlf_file_name

  (optional) Specifies the WLF files to open for analysis. The argument can be a single file or a list of files. A list of file names must be enclosed in curly braces {}. The name of the wlf file must be exactly the same as that specified in the archive write command, including the pathname, if provided.

archive write

The archive write command allows you to create a debug archive file, with the file extension .dgb, that contains one or more WLF files, debug information captured from the design library, an optional connectivity debug database file, and optional HDL source files. With this archived file, you can perform post-simulation debugging in different location from that which the original simulation was run.

Syntax

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

Arguments

- <archive_name>

  Specifies the name of the archived file to be created.

- -wlf <wlf_file_name>

  Specifies the name of the WLF file to use for post-simulation analysis. <wlf_file_name> may be a list of files enclosed in curly braces {} if you want to capture more than one WLF file in the archive.

- -include_src

  (optional) Indicate if source files should be captured in the archive. This is off by default, which means no source will be in the archive.

- -dbg <dbg_file_name>

  (optional) Specifies the name of an existing debug database (.dbg) file to be included in the archive.

batch_mode

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

Syntax

batch_mode

Arguments

None

Examples

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

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

Related Topics

- Modes of Operation

bd

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

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

Syntax

bd {<**filename**> <**line_number**>}

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

Arguments

- <**filename**>

  (required when not specifying **<id_number>** or **<label>**.) A string that specifies the name of the source file in which the breakpoint is to be deleted. The filename must match the one used previously to set the breakpoint, including whether you used a full pathname or a relative name. Must be specified as the first argument to the **bd** command.

- <**line_number**>

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

- <**id_number**> | <**label**>

  (required when not specifying **<filename>**.) Specifies the identification of breakpoints using markers assigned by the bp command. Must be specified as the first argument to the **bd** command.

  <id_number — A string that specifies the identification number of the breakpoint to be deleted. The identification number is set with the **-id** argument to the bp command.

  <label> — A string that specifies the label of the breakpoint to be deleted. The label is set with the **-label** switch to the **bp** command.

Examples

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

  ```
  bd alu.vhd 127
  ```

- Delete the breakpoint with id# 5.

  ```
  bd 5
  ```

- Delete the breakpoint with the label top_bp

  ```
  bd top_bp
  ```

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

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

Related Topics

- bp
- onbreak

bookmark add wave

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

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

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

Syntax

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

Arguments

- <**label**>

  (required) A string that specifies the name for the bookmark. Must be specified as the first argument to the **bookmark add wave** command.

- <range_start> [<unit>]

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

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

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

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

- <range_end> [<unit>]

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

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

- <topindex>

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

Examples

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

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

Related Topics

- bookmark delete wave
- bookmark goto wave
- bookmark list wave
- write format

bookmark delete wave

This command deletes bookmarks from the specified Wave window.

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

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

Syntax

bookmark delete wave {<**label**> | **-all**}

Arguments

- <**label**> | **-all**

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

  <label> — Specifies the name of the bookmark to delete.

  -all — Specifies that all bookmarks in the window be deleted.

Examples

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

```
bookmark delete wave foo
```

Related Topics

- bookmark add wave
- bookmark goto wave
- bookmark list wave
- write format

bookmark goto wave

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

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

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

Syntax

bookmark goto wave <**label**>

Arguments

- <**label**>

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

Related Topics

- bookmark add wave
- bookmark delete wave
- bookmark list wave
- write format

bookmark list wave

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

Syntax

bookmark list wave

Related Topics

- bookmark add wave
- bookmark delete wave
- bookmark goto wave
- write format

bp

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

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

Syntax

Setting an HDL breakpoint

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

Querying a breakpoint

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

Reporting all breakpoints

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

bp

returns:

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

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

Arguments

- <**filename**>

  (required to set an HDL breakpoint) Specifies the name of the source file in which to set the breakpoint. Must be specified as the first argument to the **bp** command.

- <**line_number**>

  (required to set an HDL breakpoint) Specifies the line number where the breakpoint is to be set. Must be specified as the second argument to the **bp** command.

- -appendinst

  (optional) When specifying multiple breakpoints with **-inst**, append each instance-path condition to the earlier condition. This overrides the default behavior, in which each condition overwrites the previous one.

- -disable

  (optional) Sets the breakpoint to a disabled state. You can enable the breakpoint later using the **enablebp** command. This command enables breakpoints by default.

- <command>…

  (optional, must be specified as the final argument) Specifies one or more commands that are to be executed at the breakpoint. You must separate multiple commands with semicolons (;) or place them on multiple lines. Braces are required only if the string contains spaces.

Note

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

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

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

- -cond "<condition_expression>"

  (optional) Specifies one or more conditions that determine whether the breakpoint is hit.

  "<condition_expression>" — A conditional expression that results in a true/false value. You must enclose the condition expression within braces ({}) or quotation marks (“ ”) when the expression makes use of spaces. Refer to the note below when setting breakpoints in the GUI.

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

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

Note

You can also specify this expression by choosing **Tools > Breakpoints...** from the main menu and entering the expression in the **Breakpoint Condition** field of the **Modify Breakpoints** dialog box. Do not enclose the condition expression in quotation marks (“ ”) or braces ({}).

The condition expression can use the following operators:

Operation	Operator Syntax
equals	==, =

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Note

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

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

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

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

Note

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

- -query <filename> [<line_number>]

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

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

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

  The output contains six fields of information. For example:

  bp -query top.vhd 70

  returns

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

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

    0 — Breakpoint does not exit.

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

    0 — Breakpoint is not enabled.

1 — Breakpoint is enabled.

Examples

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

  bp

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

  bp alu.vhd 147

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

  bp alu.vhd 147 {do macro.do}

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

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

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

  bp test.vhd 14 -cond {clk=1 AND prdy=0}

- Set a breakpoint with the label top_bp

  bp top.vhd 14 -label top_bp

- Set a breakpoint for line 15 of a.vhd, but only for the instance a2:

  bp a.vhd 15 -inst "/top/a2"

- Set multiple breakpoints in the source file *test.vhd* at line 14. The second instance will overwrite the conditions of the first.

  bp test.vhd 14 -inst /test/inst1 -inst /test/inst2

- Set multiple breakpoints at line 14. The second instance will append its conditions to the first.

  bp test.vhd 14 -inst /test/inst1 -inst /test/inst2 -appendinst

- Set a breakpoint for a specific variable of a particular class type:

  set x [examine -handle my_class_var]

  bp top.sv 15 -cond {this == $x}

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

  bp -query testadd.vhd

- List details about the breakpoint on line 48.

  bp -query testadd.vhd 48

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

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

Note

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

Related Topics

- bd
- disablebp
- enablebp
- Editing File-Line Breakpoints
- onbreak
- when

call

This command calls SystemVerilog static functions and class functions directly from the vsim command line in live simulation mode. Tasks are not supported.

Function return values are returned to the vsim shell as a Tcl string. If the function returns a class reference, the class instance ID is returned.

Syntax

Calling a function

call <**pathToFunction**> [classInstancePath] [functionArg0] [functionArg1] ...

Arguments

- pathToFunction

  (required) The name of a function. The function name may be qualified in one of three ways:

  1. By specifying the path to the function declaration, through the structural hierarchy, or declaration hierarchy. Hierarchical paths must be specified as a full path to a function or a function that exists relative to the current context (as shown in the Structure window, or returned by the environment command).
  2. By specifying a class instance hierarchical path.
  3. By specifying a class instance id string.

- classInstancePath

  (optional) Must be specified if the function path is a declaration path and the function is a non-static class function. Conversely, the class instance path name must not be specified if the given function path is a class instance variable reference or a class instance name in the format @<class_type>@nnn. This is because the class instance information can be extracted from the pathname itself.

- functionArg0 functionArg1 ...

  (optional) All arguments required by the function are specified in a space separated list in declaration order. If a function has default arguments, the arguments may be omitted from the command line provided that the arguments occur at the end of the declaration list. Function input arguments can be constant values including integers, enumerated values, and strings. A string containing spaces or special characters must be enclosed in double quotes (" ") or braces ({ }) or Tcl will try to interpret the string. For example: "my string" or {my string}. Arguments can also be design objects. Class references can be arguments, specified by either their design instance path or class instance id string. If a function has type inout, out, or ref arguments, suitable user design objects must be passed in as arguments. Any passed in argument will first be tested to determine if it is an appropriate constant value. If it is not, then the argument will be tested to determine if it is a design object. Consequently, where there is ambiguity between a constant string and the name of a design object, the

constant will be given precedence. If in this case the design object is desired, the full hierarchical path to the object can be supplied to differentiate it from the constant string.

Examples

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

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

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

  call /utop/tmyfcns.f_intint 37

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

  call /utop/tmyfcns.f_voidclasscolor /utop/tmyfcns

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

  call /utop/tmyfcns.f_voidclasscolor @myType@3

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

  call @myType@543.get_full_name

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

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

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

  VSIM> call @uvm_sequencer__3@3.get_full_name

  Returns:

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

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

  VSIM> call moduleX.who_am_i

  Returns:

  ```
  # @myClassX@4
  ```

cd

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

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

Syntax

cd [<dir>]

Arguments

- <dir>

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

change

This command modifies the value of a:

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

Syntax

change **<variable> <value>**

Arguments

- <**variable**>

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

  Wildcards are not permitted.

  The following sections list supported objects:

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

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

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

- <**value**>

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

Note

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

Examples

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

  ```
  change count 16#FFFF
  ```

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

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

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

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

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

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

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

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

Related Topics

- force

classinfo

This command allows the user to access information about class types and instances available in the simulation. You can view, find and report statistics about class instances.

The following table summarizes the class viewing options:

Command	Description
classinfo find	Reports on the current state of a particular class instance, whether it exists, has been destroyed, or has not yet been created.
classinfo instances	Displays the list of class items in the specified class.
classinfo report	Prints a detailed report of a set of classes and their usage.
classinfo stats	Displays statistics about the number of class types and instances.
classinfo types	Displays the list of classes that match or do not match the specified pattern.

Syntax

classinfo find [-tcl] [-o <outfile>] **class-instance-name**

classinfo instances [-tcl] [-o <outfile>] **classname**

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

classinfo stats [-tcl] [-o <outfile>]

classinfo types [-tcl] [-n] [-o <outfile>] [-x] **pattern**

Arguments

- -c [fntpc]

  Display the report columns in the specified order in a report. The default is to display all columns in the following order: Full Path, Class Name, Total, Peak, Current. You can specify one or more columns in any order.

  f — The Full Path column displays the full relative path name.

  n — The Class Name column displays the name of the class instance.

  t — The Total column displays the total number of instances of the named class.

  p — The Peak column displays the maximum number of instances of the named class that existed simultanously at any time in the simulation.

  c — The Current column displays the current number of instances of the named class.

- class-instance-name

  (required for classinfo find) Name of the class item in the following format @<name>@#.

- classname

  (required for classinfo types) Name of the class or the full path of the class type.

- -m <maxout>

  Display the specified number of lines of the report.

  <maxout> — Any non-negative integer.

- -n

  Returns class names only. Does not include the path unless required to resolve name ambiguity.

- -o <outfile>

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

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

- pattern

  A standard TCL glob expression used as a search string.

- -sort [a | d][f | n | t | p | c]

  Specifies whether the report information is sorted in ascending or descending order and which column to sort by. Only one column can be specified for sorting.

  a — Sort the entries in ascending order.

  d — Sort the entries in descending order.

  f — Sort by the Full Path column

  n— Sort by the Class Name column

  t — Sort by the Total column

  p — Sort by the Peak column

  c — Sort by the Current column

- -tcl

  Returns a tcl list instead of formatted output.

- -x

  Display classes that do not match the pattern.

- -z

  Remove all items from the report with a total instance count of zero.

Example

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

  vsim> classinfo types -x *uvm*

  Returns:

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

```
# /mem_agent_pkg::mem_config
# /mem_agent_pkg::mem_driver
```

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

  vsim> classinfo stats

  Returns:

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

- List the current instances for the class type mem_item.

  vsim> classinfo instances mem_item

  Returns:

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

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

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

  Returns:

```
# Class Name               Total   Peak Current
# uvm_pool__11               318    315    315
# uvm_event                  286     55     52
# uvm_callback_iter__1       273      3      2
# uvm_queue__3               197     13     10
# uvm_object_string_pool__1  175     60     58
# mem_item                   140     25     23
```

- Find the class instance @mem_item@87

  VSIM> classinfo find @mem_item@87

  Returns:

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

- Find the class instance @mem_item@200

  VSIM> classinfo find @mem_item@200

Returns:

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

Related Topics

- ClassDebug modelsim.ini variable
- vsim -classdebug

configure

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

To change the default window, use the view command.

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

Syntax

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

---- List Window Arguments
[-delta [all | collapse | events | none]] [-gateduration [<duration_open>]]
[-gateexpr [<expression>]] [-usegating [off | on]] [-strobeperiod [<period>[<unit>]]]
[-strobestart [<start_time>[<unit>]]] [-usesignaltriggers [0 | 1]] [-usestrobe [0 | 1]]

---- Wave Window Arguments
[-childrowmargin [<pixels>]] [-cursorlockcolor [<color>]] [-gridauto [off | on]]
[-gridcolor [<color>]][-griddelta [<pixels>]] [-gridoffset [<time>[<unit>]]]
[-gridperiod [<time>[<unit>]]] [-namecolwidth [<width>]] [-rowmargin [<pixels>]]
[-signalnamewidth [<value>]] [-timecolor [<color>]] [-timeline [0 | 1]]
[-timelineunits [fs | ps | ns | us | ms | sec | min | hr]] [-valuecolwidth [<width>]]
[-vectorcolor [<color>]] [-waveselectcolor [<color>]] [-waveselectenable [0 | 1]]

Description

The command works in three modes:

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

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

Arguments

- **list | wave**

  (required) Controls the widget to configure. Must be specified as the first argument to the **configure** command.

  list — Specifies the List widget.

  wave — Specifies the Wave widget.

- <option> <value>

  -bg <color> — (optional) Specifies the window background color.

  -fg <color> — (optional) Specifies the window foreground color.

  -selectbackground <color> — (optional) Specifies the window background color when selected.

  -selectforeground <color> — (optional) Specifies the window foreground color when selected.

  -font <font> — (optional) Specifies the font used in the widget.

  -height <pixels> — (optional) Specifies the height in pixels of each row. .

Arguments, List window only

- -delta [all | collapse | events | none]

  (optional) Specifies how information is displayed in the delta column. To use **-delta**, **-usesignaltriggers** must be set to 1 (on).

  all — Displays a new line for each time step on which objects change.

  collapse — Displays the final value for each time step.

  events — Displays an "event" column rather than a "delta" column and sorts List window data by event.

  none — Turns off the display of the delta column.

- -gateduration [<duration_open>]

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

  <duration_open> — Any non-negative integer where the default is 0 (values are not displayed).

- -gateexpr [<expression>]

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

  <expression> — An expression.

- -usegating [off | on]

  (optional) Enables triggers to be gated on or off by an overriding expression. (Use the **-gatexpr** argument to specify the expression.) Refer to "Using Gating Expressions to Control Triggering" for additional information on using gating with triggers.

  off — (default) Triggers are gated off (a value of 0).

on — Triggers are gated on (a value of 1).

- -strobeperiod [<period>[<unit>]]

  (optional) Specifies the period of the list strobe.

  <period> — Any non-negative integer.

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

- -strobestart [<start_time>[<unit>]]

  (optional) Specifies the start time of the list strobe.

  <start_time> — Any non-negative integer.

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

- -usesignaltriggers [0 | 1]

  (optional) Specifies whether or not signals are to be used as triggers.

  0 — Signals are not used as triggers

  1 — Signals are used as triggers

- -usestrobe [0 | 1]

  (optional) Specifies whether or not a strobe is used as a trigger.

  0 — Strobe is not used to trigger.

  1 — Strobe is used to trigger.

Arguments, Wave window only

- -childrowmargin [<pixels>]

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

  <pixels> — Any non-negative integer where the default is 2.

- -cursorlockcolor [<color>]

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

  <color> — Any Tcl color where the default is red.

- -gridauto [off | on]

  (optional) Controls the grid period when in simulation time mode.

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

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

- -gridcolor [<color>]

  (optional) Specifies the background grid color. Related Tcl variable is PrefWave(gridColor).

  <color> — Any color where the default is grey50.

- -griddelta [<pixels>]

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

  <pixels> — Any non-negative integer where the default is 40.

- -gridoffset [<time>[<unit>]]

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

  <time> — Any non-negative integer where the default is 0.

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

- -gridperiod [<time>[<unit>]]

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

  <time> — Any non-negative integer where the default is 1.

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

- -namecolwidth [<width>]

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

  <width> — Any non-negative integer where the default is 150.

- -rowmargin [<pixels>]

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

  <pixels> — Any non-negative integer where the default is 4.

- -signalnamewidth [<value>]

  (optional) Controls the number of hierarchical regions displayed as part of a signal name shown in the pathname pane. Related Tcl variable is PrefWave(SignalNameWidth). Can also be set with the WaveSignalNameWidth variable in the *modelsim.ini* file.

<value> — Any non-negative integer where the default is 0 (display the full path. For example,1 displays only the leaf path element, 2 displays the last two path elements, and so on.

- -timecolor [<color>]

  (optional) Specifies the time axis color. Related Tcl variable is PrefWave(timeColor).

  <color> — Any color where the default is green.

- -timeline [0 | 1]

  (optional) Specifies whether the horizontal axis displays simulation time or grid period count. Related Tcl variable is PrefWave(timeline).

  0 — (default) Simulation time is displayed.

  1 — Grid period count is displayed.

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

  (optional) Specifies units for timeline display. Does not affect the currently-defined simulation time.

  fs — femtosecond (10^{-15} seconds)

  ps — picosecond (10^{-12} seconds)

  ns — nanosecond (10^{-9} seconds) (default)

  us — microsecond (10^{-6} seconds)

  ms — millisecond (10^{-3} seconds)

  sec — second

  min — minute (60 seconds)

  hr — hour (3600 seconds)

- -valuecolwidth [<width>]

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

  <width> — Any non-negative integer where the default is 100.

- -vectorcolor [<color>]

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

  <color> — Any color where the default is #b3ffb3.

- -waveselectcolor [<color>]

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

  <color> — Any color where the default is grey30.

- -waveselectenable [0 | 1]

  (optional) Specifies whether the waveform background highlights when an object is selected. Related Tcl variable is PrefWave(waveSelectEnabled).

  0 — (default) Highlighting is disabled.

  1 — Highlighting is enabled.

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

Examples

- Display the current value of the strobeperiod attribute.

  config list -strobeperiod

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

  config list -strobeperiod {50 ns} -strobestart 0 -usestrobe 1

- Set the wave vector color to blue.

  config wave -vectorcolor blue

- Set the display in the current Wave window to show only the leaf path of each signal.

  config wave -signalnamewidth 1

Related Topics

- view
- Simulator GUI Preferences

dataset alias

This command maps an alternate name (alias) to an open dataset. A dataset can have any number of aliases, but all dataset names and aliases must be unique even when more than one dataset is open. Aliases are not saved to the *.wlf* file and must be remapped if the dataset is closed and then re-opened.

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

Syntax

dataset alias **<dataset_name>** [<alias_name>]

Arguments

- <dataset_name>

  (required) Specifies a dataset name or currently assigned dataset alias. Must be specified as the first argument to the **dataset alias** command. Returns a list of all aliases mapped to the specified dataset file when specified without <dataset_alias>.

- <alias_name>

  (optional) Specifies string to assign to the dataset as an alias. Wildcard characters are permitted.

Examples

Assign the alias name “bar” to the dataset named “gold.”

```
dataset alias gold bar
```

Related Topics

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

dataset clear

This command applies only to WLF based simulation datasets. It has no effect on coverage (UCDB) datasets. All event data is removed from the current simulation WLF file, while retaining all currently logged signals. Subsequent run commands will continue to accumulate data in the WLF file.

If the command is executed when no design is loaded then the error: “Dataset not found:sim” is returned. If the command is executed when a design is loaded, then the “sim:” dataset is cleared, irrespective of which dataset is currently set. Clearing the dataset will clear any open wave window based on the “sim:”.

Syntax

dataset clear

Examples

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

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

Related Topics

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

dataset close

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

Syntax

dataset close {<dataset_name> | -all}

Arguments

- <dataset_name> | -all

  (required) Closes active dataset(s).

  <dataset_name> — Specifies the name of the dataset or alias you wish to close.

  -all — Closes all open datasets and the simulation.

Related Topics

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

dataset config

This command configures WLF parameters for an open dataset and all aliases mapped to that dataset. It has no effect on coverage datasets (UCDB).

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

Syntax

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

Arguments

- <dataset_name>

  (required) Specifies a open dataset or dataset alias you wish to configure. Must be specified as the first argument to the **dataset config** command.

- -wlfcachesize [<n>]

  (optional) Sets the size, in megabytes, of the WLF reader cache. Does not affect the WLF write cache.

  <n> — Any non-negative integer, in MB where the default is 256.

  If you do not specify a value for <n>, this switch returns the size, in megabytes, of the WLF reader cache.

- -wlfdeleteonquit [0 | 1]

  (optional) Deletes the WLF file automatically when the simulation exits. Valid for the current simulation dataset only.

  0 — Disabled (default)

  1 — Enabled

  If you do not specify an argument, this switch returns the current setting for the switch.

- -wlfopt [0 | 1]

  (optional) Optimizes the display of waveforms in the Wave window.

  0 — Disabled

  1 — Enabled (default)

  If you do not specify an argument, this switch returns the current setting for the switch.

Examples

Set the size of the WLF reader cache for the dataset “gold” to 512 MB.

```
dataset config gold -wlfcachesize 512
```

Related Topics

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

dataset current

This command activates the specified dataset and sets the GUI context to the last selected context of the specified dataset. All context dependent GUI data is updated and all context dependent CLI commands start working with respect to the new context.

Syntax

dataset current [<dataset_name>]

Arguments

- <dataset_name>

  (optional) Specifies the dataset name or dataset alias you want to activate. If no dataset name or alias is specified, the command returns the name of the currently active dataset.

Related Topics

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

dataset info

This command reports a variety of information about a dataset.

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

Syntax

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

Arguments

- {name | file | exists}

  (required) Identifies what type of information you want reported.

  Only one option per command is allowed. The current options include:

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

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

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

  Must be specified as the first argument to the **dataset info** command.

- <dataset_name>

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

Related Topics

- dataset alias
- dataset clear
- dataset close
- dataset config
- dataset list
- dataset open
- dataset rename
- dataset restart
- dataset save
- dataset snapshot
- environment

dataset list

This command lists all active datasets.

Syntax

dataset list [-long]

Arguments

- -long

  (optional) Lists the dataset name followed by the *.wlf* file to which the dataset name is mapped.

Related Topics

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

dataset open

This command opens a WLF file (either the currently running *vsim.wlf* or a saved WLF file) and/or UCDB file (representing coverage data) and assigns it the logical name that you specify. The file may be the WLF file for a currently running simulation. To close a dataset, use dataset close.

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

Syntax

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

Arguments

- <file_name>

  (required) Specifies the WLF file or UCDB file to open as a view-mode dataset. Must be specified as the first argument to the **dataset open** command. Specify *vsim.wlf* to open the currently running WLF file.

- <dataset_name>

  (optional) Specifies a name for the open dataset. This is a name that will identify the dataset in the current session. By default the dataset prefix will be the name of the specified WLF or UCDB file.

Examples

Open the dataset file *last.wlf* and assign it the name *test*.

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

Related Topics

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

dataset rename

This command changes the name of a dataset to the new name you specify.

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

dataset rename **<dataset_name> <new_dataset_name>**

Arguments

- <dataset_name>

  Specifies the existing name of the dataset.

- <new_dataset_name>

  Specifies the new name for the dataset.

Examples

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

```
dataset rename test test2
```

Related Topics

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

dataset restart

This command unloads the specified dataset or currently active dataset and reloads the dataset using the same dataset name. The contents of Wave and other coverage windows are restored for UCDB datasets after a reload.

Syntax

dataset restart [<file_name>]

Arguments

- <file_name>

  (optional) Specifies the WLF or UCDB file to open as a view-mode or coverage mode dataset. If <filename> is not specified, the currently active dataset is restarted.

Related Topics

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

dataset save

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

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

dataset save **<dataset_name> <file_name>**

Arguments

- <dataset_name>

  (required) Specifies the name of the dataset you want to save.

- <file_name>

  (required) Specifies the name of the file to save.

Examples

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

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

Related Topics

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

dataset snapshot

This command saves data from the current WLF file (*vsim.wlf* by default) at a specified interval. It provides you with sequential or cumulative "snapshots" of your simulation data. This command does not apply to coverage datasets (UCDB).

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

Syntax

dataset snapshot [-dir <directory>] [-disable] [**-enable**] [-file <file_name>] [-filemode {overwrite | increment}] [-mode {cumulative | sequential}] [-report] [-reset] **{-size <file_size> | -time <n> [<unit>]}**

Arguments

- -dir <directory>

  (optional) Specifies a directory into which the files should be saved. Either absolute or relative paths may be used. Default is to save to the current working directory.

- -disable

  (optional) Turns snapshotting off. All dataset snapshot settings from the current simulation are stored in memory. All other options are ignored after you specify **-disable**.

- -enable

  (optional) Turns snapshotting on. Restores dataset snapshot settings from memory or from a saved dataset. (default)

- -file <file_name>

  (optional) Specifies the name of the file to save snapshot data.

  <file_name> — A specified file name where the default is *vsim_snapshot.wlf*. The suffix *.wlf* will be appended to specified filename and, possibly, an incrementing suffix.

  When the duration of the simulation run is not a multiple of the interval specified by -size or -time, the incomplete portion is saved in the file *vsim.wlf*.

- -filemode {overwrite | increment}

  (optional) Specifies whether to overwrite the snapshot file each time a snapshot occurs.

  overwrite — (default)

  increment — A new file is created for each snapshot. An incrementing suffix (1 to n) is added to each new file (for example, *vsim_snapshot_1.wlf*).

- -mode {cumulative | sequential}

  (optional) Specifies whether to keep all data from the time signals are first logged.

  cumulative — (default)

  sequential — The current WLF file is cleared every time a snapshot is taken.

- -report

  (optional) Lists current snapshot settings in the Transcript window. All other options are ignored if you specify **-report**.

- -reset

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

- -size <file_size>

  (Required if -time is not specified.) Specifies that a snapshot occurs based on WLF file size. Must be specified as the final argument to the **dataset snapshot** command.

  <file_size> — Size of WLF file in MB.

- -time <n> [<unit>]

  (Required if -size is not specified.) Specifies that a snapshot occurs based on simulation time. Must be specified as the final argument to the **dataset snapshot** command.

  <n> — Any positive integer.

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

Examples

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

  ```
  dataset snapshot -size 10
  ```

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

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

- Assuming simulator time units are ps, this command saves a file called *gold_<n>.wlf* every 1000000 ps. If you run the simulation for 3000000 ps, three files are saved: *gold_1.wlf* with data from 0 to 1000000 ps, *gold_2.wlf* with data from 1000000 to 2000000, and *gold_3.wlf* with data from 2000000 to 3000000.

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

  Because this example sets the time interval to 1000000 ps, if you run the simulation for 3500000 ps, a file containing the data from 3000000 to 3500000 ps is saved as *vsim.wlf* (default).

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

dataset snapshot -reset -time 10000

Related Topics

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

delete

This command removes objects from either the List or Wave window. Arguments to this command are order dependent.

Syntax

delete **list** [-window <wname>] **<object_name>**

delete **wave** [-window <wname>] **<object_name>**

Arguments

- list

  Specifies the target is a list window.

- wave

  Specifies the target is a wave window.

- -window <wname>

  (optional) Specifies the name of the List or Wave window to target for the **delete** command. (The view command allows you to create more than one List or Wave window.) If no window is specified, the default window is used; the default window is determined by the most recent invocation of the view command and has " - Default" appended to the name.

- <object_name>...

  (required) Specifies the name of an object. Must match an object name used in an add list or add wave command. Multiple object names are specified as a space separated list. Wildcard characters are allowed. Must be specified as the final argument to the **delete list and delete wave** commands.

Examples

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

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

- Remove all objects beginning with the string /test from the Wave window.

  ```
  delete wave /test*
  ```

Related Topics

- add list
- add wave
- Wildcard Characters

describe

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

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

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

Syntax

describe **<name>...**

Arguments

- <name>...

  (required) The name of an HDL object for which you want a description. Multiple object names are specified as a space separated list. Wildcard characters are allowed. HDL object names can be relative or full hierarchical names.

Examples

- Print the types of the three specified signals.

  ```
  describe clk prw prdy
  ```

Related Topics

- add list
- add wave
- Wildcard Characters

disablebp

This command turns off breakpoints and when commands. To turn on breakpoints or when commands again, use the enablebp command.

Syntax

disablebp [<id#> | <label>]

Arguments

- <id#>

  (optional) Specifies the ID number of a breakpoint or **when** statement to disable.

- <label>

  (optional) Specifies the label name of a breakpoint or **when** statement to disable.

If you do not specify either of these arguments, all breakpoints and when statements are disabled.

Use the bp command with no arguments to find labels and ID numbers for all breakpoints in the current simulation. Use the when command with no arguments to find labels and ID numbers of all when statements in the current simulation.

Note

Id numbers for breakpoints and when statements are assigned from the same pool. Even if you have not specified a given id number for a breakpoint, that number may still be used for a when command.

Related Topics

- bd
- bp
- enablebp
- onbreak
- resume
- when

do

This command executes the commands contained in a macro file.

A macro file can have any name and extension. An error encountered during the execution of a macro file causes its execution to be interrupted, unless an onerror command or the OnErrorDefaultAction Tcl variable is specified with the resume command. The onbreak command is used to take action with source code breakpoint cases.

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

Syntax

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

Arguments

- <filename>

  (required) Specifies the name of the macro file to be executed. The name can be a pathname or a relative file name. Pathnames are relative to the current working directory. Must be specified as the first argument to the **do** command.

  If the do command is executed from another macro file, pathnames are relative to the directory of the calling macro file. This allows groups of macro files to be stored in a separate sub-directory.

- <parameter_value>...

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

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

Note

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

Examples

- Execute the file *macros/stimulus and* pass the parameter value 100 to $1 in the macro file.

  ```
  do macros/stimulus 100
  ```

  Where the macro file *testfile* contains the line

  ```
  bp $1 $2
  ```

  place a breakpoint in the source file named *design.vhd* at line 127.

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

Related Topics

- Tcl and Macros (DO Files)
- Modes of Operation
- Using a Startup File
- DOPATH variable
- Saving a Transcript File as a Macro (DO file)

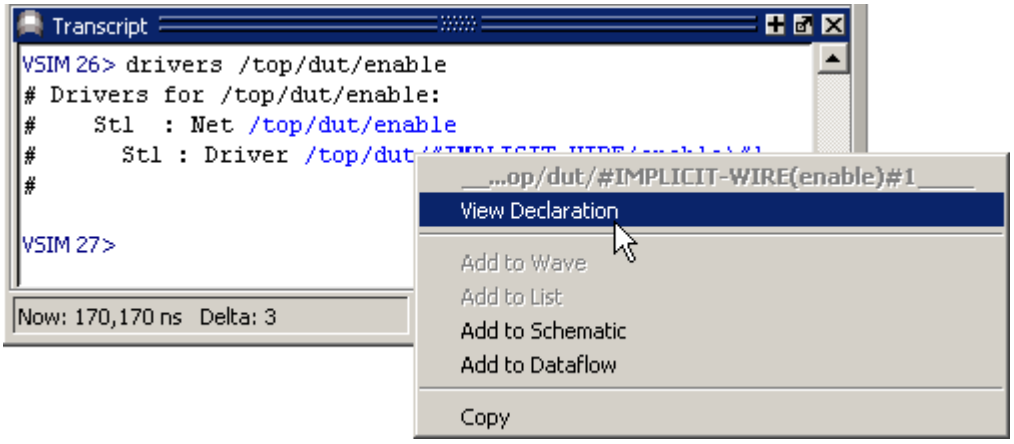
drivers

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

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

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

Figure 2-1. drivers Command Results in Transcript



Syntax

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

Arguments

- <object_name>

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

- -source

  (optional) Returns the source file name and line number for each driver of the specified signal or net. If the source location cannot be determined, the value n/a is returned for that driver.

Example

drivers /top/dut/pkt_cnt(4)

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

In some cases, the output may supply a strength value similar to 630 or 52x, which indicates an ambiguous verilog strength.

Related Topics

- readers
- Verilog Language Reference Manual Std 1365-2005 section 7.10.2 "Ambiguous strengths: sources and combinations"

dumplog64

This command dumps the contents of the specified WLF file in a readable format to stdout.

The WLF file cannot be opened for writing in a simulation when you use this command. This command cannot be used in a DO file.

Syntax

dumplog64 **<filename>**

Arguments

- <filename>

  (required) The name of the WLF file to be read.

echo

This command displays a specified message in the Transcript window.

Syntax

echo **[<text_string>]**

Arguments

- <text_string>

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

Examples

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

  echo “The time is $now ns.”

  returns the message:

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

- If the quotes are omitted:

  echo The time is $now ns.

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

  ```
  The time is $now ns.”
  ```

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

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

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

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

edit

This command invokes the editor specified by the EDITOR environment variable. By default, the specified filename will open in the Source window.

Syntax

edit [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the file to edit. If the <filename> argument is omitted, the editor opens the current source file. If you specify a non-existent filename, it will open a new file. Either absolute or relative paths may be used.

Related Topics

- notepad
- EDITOR environment variable

enablebp

This command turns on breakpoints and when commands that were previously disabled.

Syntax

enablebp [<id#> | <label>]

Arguments

- <id#>

  (optional) Specifies a breakpoint ID number or **when** statement to enable.

- <label>

  (optional) Specifies the label name of a breakpoint or **when** statement to enable.

If you do not specify either of these arguments, all breakpoints are enabled.

Use the bp command with no arguments to find labels and ID numbers for all breakpoints in the current simulation. Use the when command with no arguments to find labels and ID numbers of all when statements in the current simulation.

Related Topics

- bd
- bp
- disablebp
- onbreak
- resume
- when

encoding

This command translates between the 16-bit Unicode characters used in Tcl strings and a named encoding, such as Shift-JIS. There are four encoding commands used to work with the encoding of your character representations in the GUI.

- encoding convertfrom — Convert a string from the named encoding to Unicode.
- encoding convertto — Convert a string to the named encoding from Unicode.
- encoding names — Returns a list of all valid encoding names.
- encoding system — Changes the current system encoding to a named encoding. If a new encoding is omitted the command returns the current system encoding. The system encoding is used whenever Tcl passes strings to system calls.

Syntax

encoding convertfrom <encoding_name> <string>

encoding convertto <encoding_name> <string>

encoding names

encoding system <encoding_name>

Arguments

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

environment

This command has two forms, environment and env. It allows you to display or change the current dataset and region/signal environment.

Syntax

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

Arguments

- -dataset

  (optional) Displays the specified environment pathname *with* a dataset prefix. Dataset prefixes are displayed by default.

- -nodataset

  (optional) Displays the specified environment pathname without a dataset prefix.

- <pathname>

  (optional) Specifies a new pathname for the region/signal environment.

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

- -forward

  (optional) Displays the next environment in your history of visited environments.

- -back

  (optional) Displays the previous environment in your history of visited environments.

Examples

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

  env

- Change to another dataset but retain the currently selected context.

  env test:

- Change all unlocked windows to the context "test:/top/foo".

  env test:/top/foo

- Move down two levels in the design hierarchy.

  env blk1/u2

- Move to the top level of the design hierarchy.

  env /

Related Topics

- Refer to Object Name Syntax for information on specifying pathnames.
- Refer to Setting your Context by Navigating Source Files for more information about -forward and -back.

examine

This command has two forms, examine and exa. It examines one or more objects and displays current values (or the values at a specified previous time) in the Transcript window.

It can also compute the value of an expression of one or more objects.

The following objects can be examined:

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

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

To compute an expression, use the **-expr** option. The **-expr** and the **-time** options may be used together.

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

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

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

See Design Object Names for more information on specifying names.

Syntax

examine **<name>…** [-delta <delta>] [-env <path>] [-event <time>] [-handle] {[-in] [-out] [-inout] | [-ports]} [-internal] [-maxlen <integer>] [-expr <expression>] [-name]

[-<radix_type>] [-radix <type>] [-radixenumnumeric | -radixenumsymbolic] [-showbase] [-time <time>] [-value]

Arguments

- <name>…

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

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

  exa line__36/i

- -delta <delta>

  (optional) Specifies a simulation cycle at the specified time step from which to fetch the value, where the default is to use the last delta of the time step. You must log the objects to be examined using the add list, add wave, or log command for the examine command to be able return a value for a requested delta.

  <delta> — Any non-negative integer.

- -env <path>

  (optional) Specifies a path in which to look for an object name.

  <path> — The specified path to a object.

- -event <time>

  (optional) Specifies a simulation cycle at the specified event time from which to fetch the value. The event <time> refers to the event time relative to events for all signals in the objects dataset at the specified time. You must log the objects to be examined using the add list, add wave, or log command for the examine command to be able return a value for a requested event.

- -expr <expression>

  (optional) Specifies an expression to be examined. You must log the expression using the add list, add wave, or log command for the examine command to return a value for a specified expression. The expression is evaluated at the current time simulation. If you also specify the **-time** argument, the expression will be evaluated at the specified time. It is not necessary to specify <name> when using this argument. See GUI_expression_format for the format of the expression.

  <expression> — Specifies an expression enclosed in braces ({}).

- -handle

  (optional) Returns the memory address of the specified <name>. You can use this value as a tag when analyzing the simulation. This value also appears as the title of a box in the Watch window. This option will not return any value if you are in -view mode.

- -in

  (optional) Specifies that <name> include ports of mode IN.

- -out

  (optional) Specifies that <name> include ports of mode OUT.

- -inout

  (optional) Specifies that <name> include ports of mode INOUT.

- -internal

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

- -maxlen <*integer*>

  (optional) Specifies the maximum number of characters in the output of the command.

  <integer> — Any non-negative integer where 0 is unlimited.

- -ports

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

- -name

  (optional) Displays object name(s) and value(s). Related switch is -value.

- -<radix_type>

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

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

- -radix <type>

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

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

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

- -radixenumnumeric

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

- -radixenumsymbolic

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

- -showbase

  (optional) Display the number of bits of the vector and the radix used, where:

  binary = b
  decimal = d
  hexidecimal = h
  ASCII = a
  time = t

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

- -time <time>

  (optional) Specifies the time value between 0 and $now for which to examine the objects.

  <time> — A non negative integer where the default unit is the current time unit. If the <time> field uses a unit other than the current unit, the value and unit must be placed in curly braces. For example, the following are equivalent for ps resolution:

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

  If an expression is specified it will be evaluated at that time. The objects to be examined must be logged via the add list, add wave, or log command in order for the examine command to be able to return a value for a requested time.

- -value

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

Examples

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

  examine /top/bus1

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

  {rega[16]}

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

  examine {foo[20:22]}

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

  examine {/top/\My extended id\ }

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

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

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

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

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

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

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

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

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

  examine -hex [find *]
  ```

Related Topics

- Design Object Names
- Wildcard Characters
- DefaultRadix *modelsim.ini* variable

exit

This command exits the simulator and the ModelSim application.

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

Syntax

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

Argument

- -force

  (optional) Quits without asking for confirmation. If this argument is omitted, ModelSim asks you for confirmation before exiting. You can also use -f as an alias for this switch.

- -code <integer>

  (optional) Quits the simulation and issues an exit code.

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

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

Examples

You can use exit -code to instruct a vmake command to exit when it encounters an assertion error. The onbreak command can specify commands to be executed upon an assert failure of sufficient severity, after which the simulator can be made to return an exit status. This is shown in the following example:

```
set broken 0
onbreak {
  set broken 88
  resume
}
run -all
if { $broken } {
  puts "failure -- exit status $broken"
  exit -code $broken
} else {
  puts "success"
}
quit -f
```

The resume command gives control back to the commands following the run -all to handle the condition appropriately.

find

This command locates objects by type and name. Arguments to the command are grouped by object type:

- Arguments for nets and signals
- Arguments for instances and blocks
- Arguments for virtuals
- Arguments for classes
- Arguments for objects

Syntax

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

find instances | blocks **{<object_name> …** | -bydu <design_unit> | -file <file_name>**}** [-recursive] [-nodu]

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

find classes [<class_name>]

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

Arguments for nets and signals

When searching for nets and signals, the find command returns the full pathname of all nets, signals, registers, variables, and named events that match the name specification.

- <object_name> …

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

- -in

  (optional) Specifies that the scope of the search is to include ports of mode IN.

- -inout

  (optional) Specifies that the scope of the search is to include ports of mode INOUT.

- -internal

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

- -nofilter

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

- -out

  (optional) Specifies that the scope of the search is to include ports of mode OUT.

- -ports

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

- -recursive

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

Arguments for instances and blocks

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

- -bydu <design_unit>

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

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

- -file <file_name>

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

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

- <object_name> …

  Specifies the name of an instance or block for which you want to search. Multiple instances and wildcard characters are allowed. Mutually exclusive with -file and -bydu.

- -recursive

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

- -nodu

  (optional) Removes the "du" string from the names of design units found with -bydu argument.

Arguments for virtuals

When searching for virtuals, all optional arguments must be specified before any object names.

- <object_name> …

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

- -kind <kind>

  (optional) Specifies the kind of virtual object for which you want to search.

<kind> — A virtual object of one of the following kinds:

- designs
- explicits
- functions
- implicits
- signals.

- -unsaved

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

Arguments for classes

- <class_name>

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

Arguments for objects

- -class <class_name>

  (optional) Restricts the search to objects whose most-specific class is **class_name**.

- -isa <class_name>

  (optional) Restricts the search to those objects that have **class_name** anywhere in their heritage.

- <object_name>

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

Description

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

- If the name does not include a dataset name, then the current dataset is used.
- If the name does not start with a path separator, then the current context is used.
- If the name is a path separator followed by a name that is not the name of a top-level design unit, then the first top-level design unit in the design is used.
- For a relative name containing a hierarchical path, if the first object name cannot be found in the current context, then an upward search is done up to the top of the design hierarchy to look for a matching object name.

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

See Design Object Names for more information on specifying names.

Examples

- Find all signals in the entire design.

  find signals -r /*

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

  find nets -in /top/xy*

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

  find signals -r u1/u2/cl*

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

  find signals {s[1]}

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

  find signals {s[123]}

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

  find signals s(1)

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

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

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

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

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

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

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

```
# Search for all instances with u1 in path
set pattern_match "*u1*" ;

# Get the list of instance paths
set inst_list [find instances -r *]  ;

# Initialize an empty list to strip off the architecture names
set ilist [list]  ;

foreach inst $inst_list {
set ipath [lindex $inst 0]
 if {[string match $pattern_match $ipath]} {
 lappend ilist $ipath
 }
}
# At this point, ilist contains the list of instances only--
# no architecture names
#
# Begin sorting list
set ilist [lsort -dictionary $ilist]

# Open a file to write out the list
set fhandle [open "instancelist.txt" w]
foreach inst $ilist {
 # Print instance path, one per line
 puts $fhandle $inst
}

# Close the file, done.
close $fhandle ;
```

Related Topics

- Design Object Names
- Wildcard Characters

find connections

This command returns the set of nets that are electrically equivalent to a specified net.

Syntax

find connections <net>

- <net> — (required) A net in the design. Returns a list of nets connected to the specified net. For example:

  find connections /top/p/strb

  returns:

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

find infiles

This command searches for a string in the specified file(s) and prints the results to the Transcript window. The results are individually hotlinked and will open the file and display the location of the string.

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

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

find infiles **<string_pattern> <file>...**

Arguments

- <string_pattern>

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

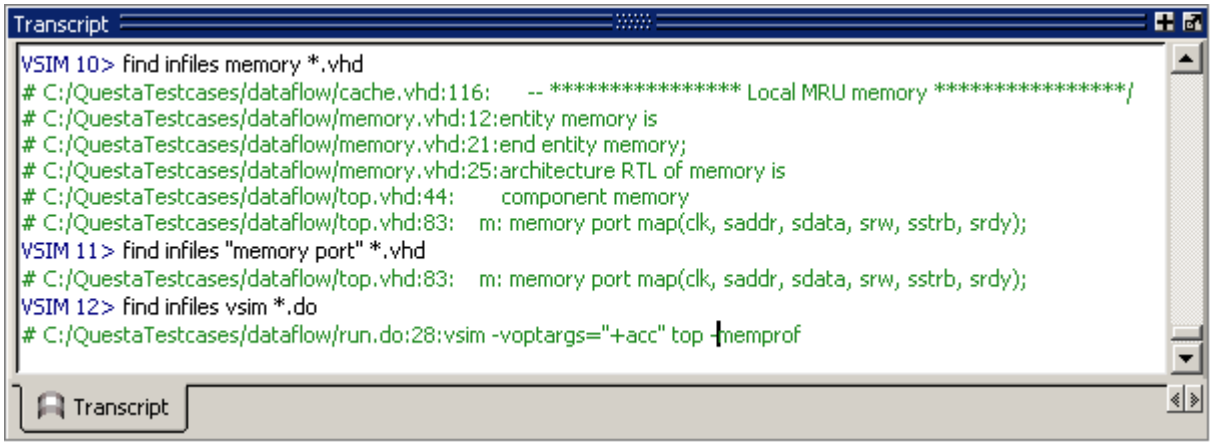
- <file>...

  (required) The file(s) to search. You can use Tcl regular expression wildcards to further restrict the search capability.

Example

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

Figure 2-2. find infiles Example



find insource

This command searches for a string in the source files for the current design and prints the results to the Transcript window. The results are hotlinked individually and will open the file and display the location of the string.

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

Syntax

find insource **<pattern>** [-exact | -glob | -regex] [-inline] [-nocase]

Arguments

- <pattern>

  (required) The string you are searching for. You can use regular expression wildcards to further restrict the search. You must enclose <pattern> in quotes (“ “) if it includes spaces. You must specify the <pattern> at the end of the command line; any switches specified after <pattern> will not be registered.

- -exact | -glob | -regex

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

  -exact — Indicates that no characters have special meaning, thus disabling wildcard features.

  -glob — (default) Allows glob-style wildcard characters. For more information refer to the Tcl documentation:

  Help > Tcl Man Pages

  Select “Tcl Commands”, then “string”, then “string match”

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

  Help > Tcl Man Pages

  Select “Tcl Commands”, then “re_syntax”.

- -inline

  (optional) Returns the matches in the form of a Tcl list, which disables the hotlink feature but allows for easier post-processing.

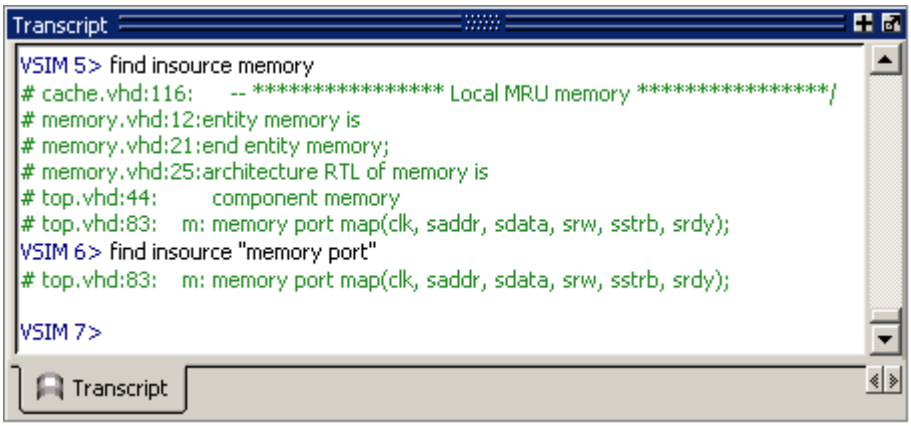
- -nocase

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

Example

Figure 2-3 shows a couple of examples of the find insource command and the results.

Figure 2-3. find insource Example



Related Topics

DISABLE_ELAB_DEBUG environment variable

force

This command allows you to apply stimulus interactively to VHDL signals, Verilog nets and registers, and SystemC boundary types.

It is possible to create a complex sequence of stimuli when the force command is included in a macro file.

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

You can force:

- VHDL signals or parts of signals.
- Verilog nets and registers, bit-selects, part-selects, and field-selects. Refer to "Force and Release Statements in Verilog" for more information.
- "Virtual Signals" if the number of bits corresponds to the signal value.
- An alias of a VHDL signal.
- An input port that is mapped at a higher level in VHDL and mixed models.

You cannot force:

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

This command provides additional information with the -help switch.

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

Syntax

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

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

Reporting all force commands

If you specify this command without arguments, it returns a list of the most recently applied force commands and a list of forces coming from the Signal Spy signal_force() and $signal_force() calls from within VHDL, Verilog, and SystemC source code. For example, after executing:

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

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

entering:

force

returns:

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

Note

The simulator translates the relative time you specify into absolute time when the force command is executed.

Arguments

- <object_name>

  (required when forcing a value change) Specifies the name of the HDL object to be forced. A wildcard is permitted only if it matches one object. Refer to Design Object Names and Tcl Syntax and Specification of Array Bits and Slices for the full syntax of an object name. The object name must specify a scalar type or a one-dimensional array of character enumeration. You may also specify a record sub-element, an indexed array, or a sliced array, as long as the type is one of the above. Must be specified as the first argument to the **force** command.

- <value>

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

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

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

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.

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

For example:

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

- -cancel [@]<time_info>

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

  where:

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

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

- -drive

  (optional) Attaches a driver to the object and drives the specified <value> until the object is forced again or until it is unforced with the noforce command.

  This option is illegal for unresolved signals.

- -deposit

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

Note

If the **-freeze**, **-drive**, or **-deposit** options are not used, then **-freeze** is the default for unresolved objects, and **-drive** is the default for resolved objects. If you prefer **-freeze** as the default for resolved and unresolved VHDL signals, change the DefaultForceKind variable in the *modelsim.ini* file.

- -freeze

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

Note

If you prefer **-freeze** as the default for resolved and unresolved VHDL signals, change the DefaultForceKind variable in the *modelsim.ini* file.

- -repeat [@]<time_info>

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

  where:

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

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

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

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

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

- [@]<time_info>

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

  where:

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

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

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

  <time_value> — The time (either relative or absolute) to apply to <value>. Any non-negative integer. A value of zero cancels the force at the end of the current time period.

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

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

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

Note

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

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

Examples

- Reporting all recently applied force commands

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

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

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

Entering:

force

returns:

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

Note

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

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

  force input1 0

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

  force bus1(4) 1

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

  force bus1 01XZ 100 ns

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

  force bus1 16#f @200

- Force *input1* to 1 at 10 time units after the current simulation time and to 0 at 20 time units after the current simulation time. Repeat this cycle every 100 time units after the current simulation time, If the current simulation time is 100 ns, the next transition is to 1 at 110 ns and 0 at 120 ns, this pattern to start repeating at 200 ns.

  force input1 1 10, 0 20 -r 100

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

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

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

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

  So,

  force s 1 0 -cancel 0

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

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

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

- Force one bit of a record containing an array.

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

- Force a slice of an array.

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

Related Topics

- change
- DefaultForceKind
- Design Object Names
- Force and Release Statements in Verilog
- Force Command Defaults
- noforce
- Virtual Signals

formatTime

This command provides global format control for all time values displayed in the GUI. When specified without arguments, this command returns the current state of the three arguments.

Syntax

formatTime [[+|-]commas] [[+|-]nodefunits] [[+|-]bestunits]

Arguments

- [+|-]commas

  (optional) Insert commas into the time value.

  + prefix — On

  - prefix — Off. (default)

- [+|-]nodefunits

  (optional) Do not include default unit in the time.

  + prefix — On

  - prefix — Off. (default)

- [+|-]bestunits

  (optional) Use the largest unit value possible.

  + prefix — On

  - prefix — Off. (default)

Examples

- Display commas in time values.

  formatTime +commas

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

- Use largest unit value possible.

  formatTime +bestunits

  Displays 8 us instead of 8,000 ns.

help

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

Syntax

help [<command> | <topic>]

Arguments

- <command>

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

- <topic>

  (optional) Specifies a topic for which you want help. The entry is case and space sensitive. Specify one of the following six topics:

Topic	Description
commands	Lists all available commands and topics
debugging	Lists debugging commands
execution	Lists commands that control execution of your simulation.
Tcl	Lists all available Tcl commands.
Tk	Lists all available Tk commands
incrTCL	Lists all available incrTCL commands

history

This command lists the commands you have executed during the current session. History is a Tcl command. For more information, consult the Tcl Man Pages (Help > Tcl Man Pages).

Syntax

history [clear] [keep <value>]

Arguments

- clear

  (optional) Clears the history buffer.

- keep <value>

  (optional) Specifies the number of executed commands to keep in the history buffer.

  <value> — Any positive integer where the default is 50.

layout

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

The command options include:

- **layout active** – returns the current active window
- **layout current** – lists the current layout
- **layout delete** – removes the current layout from the *.modelsim* file (UNIX/Linux) or Registry (Windows)
- **layout load** – opens the specified layout
- **layout names** – lists all known layouts
- **layout normal** – minimizes the current maximized window
- **layout maximized** – return a 1 if the current layout is maximized, or a 0 if minimized
- layout restoretype — removes the list of window type(s) that will not be restored when a new layout is loaded.
- **layout save** – saves the current layout to the specified name
- layout showsuppresstypes — lists the window types that will not be restored when a new layout is loaded.
- layout suppresstype — adds the specified window type(s) to the list of types that will not be restored when a layout is reloaded.
- **layout togglezoom** – toggles the current zoom state of the active window (from minimized to maximized or maximized to minimized)
- **layout zoomactive** – maximizes the current active window
- **layout zoomwindow** – maximizes the specified window

Syntax

layout active

layout current

layout delete **<name>**

layout load **<name>**

layout names

layout normal

layout maximized

layout restoretype <window>

layout save **<name>**

layout showsuppresstypes

layout suppresstype **<window>**

layout togglezoom

layout zoomactive

layout zoomwindow **<window>**

Arguments

- <name>

  (required) Specifies the name of the layout.

- <window>

  (required) The window specification can be any format accepted by the view command. The window can be specified by its type (i.e., wave, list, objects, etc.), by the windowobj name (i.e., .main_pane.wave, .main_pain.library, etc.), or by the tab name (i.e., wave1, list3, etc.)

Related Topics

- Customizing the Simulator GUI Layout
- Configuring Default Windows for Restored Layouts

log

This command creates a wave log format (WLF) file containing simulation data for all HDL objects whose names match the provided specifications. Objects that are displayed using the add list and add wave commands are automatically recorded in the WLF file. By default the file is named *vsim.wlf* and stored in the current working directory. You can change the default name using the -wlf option of the vsim command or by setting the WLFFilename variable in the *modelsim.ini* file.

If no port mode is specified, the WLF file contains data for all objects in the selected region whose names match the object name specification.

The WLF file contains a record of all data generated for the list and wave windows during a simulation run. Reloading the WLF file restores all objects and waveforms and their complete history from the start of the logged simulation run. See dataset open for more information.

For all transaction streams created through the SCV or Verilog APIs, logging is enabled by default. A transaction is logged to the WLF file if logging is enabled at the beginning of a simulation run when the design calls ::begin_transaction() or $begin_transaction. The effective start time of the transaction (the time passed by the design as a parameter to ::begin_transaction) is irrelevant. For example, a stream could have logging disabled between T1 and T2 and **still** record a transaction in that period, through retroactive logging after time T2. A transaction is always either entirely logged or entirely ignored.

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

Note

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

Syntax

log [-howmany] {[-in] [-inout] [-out] | [-ports]} [-internal]
[-out] [-ports] [-recursive [-depth <level>]] **<object_name> …**

log -flush [<object>]

Arguments

- -depth <level>

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

  <level> — Any non-negative integer. For example, if you specify -depth 1, the command descends only one level in the hierarchy.

- -flush [<object>]

  (optional) Forced the WLF file to write all buffered region and event data to the WLF file. By default, the region and event data is buffered and periodically written to the file, as appropriate. If <object> is specified, that object is first logged and then the file is flushed.

- -howmany

  (optional) Returns an integer indicating the number of signals found.

- -in

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

- -inout

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

- -internal

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

- -out

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

- -ports

  (optional) Specifies that the scope of the search is to include all ports, IN, INOUT, and OUT.

- -recursive

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

- <object_name>

  (required) Specifies the object name that you want to log. Must be specified as the final argument to the **log** command. Multiple object names are specified as a space separated list. Wildcard characters are allowed. Note that the *WildcardFilter* Tcl preference variable identifies types to ignore when matching objects with wildcard patterns.

  By default, wildcard card logging does not log the internals of cells. Refer to the **+libcell | +nolibcell** argument of the vlog command for more information.

Examples

- Log all objects in the design.

  ```
  log -r /*
  ```

- Log all output ports in the current design unit.

  ```
  log -out *
  ```

Related Topics

- add list
- add wave
- dataset alias
- dataset clear
- dataset close
- dataset config
- dataset config
- dataset info
- dataset list
- dataset open
- dataset restart
- dataset rename
- dataset save
- dataset snapshot
- nolog
- Recording Simulation Results With Datasets
- vlog **+libcell | +nolibcell**
- Wildcard Characters

lshift

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

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

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

Syntax

lshift **<list>** [<amount>]

Arguments

- <list>

  (required) Specifies the Tcl list to target with **lshift**. Must be specified as the first argument to the **lshift** command.

- <amount>

  (optional) Specifies the number of places to shift where the default is 1.

Examples

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

Related Topics

- Refer to the Tcl man pages (**Help > Tcl Man Pages**) for details.

lsublist

This command returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern.

Arguments to this command are order dependent. Follow the order specified in the Syntax section.

Syntax

lsublist **<list> <pattern>**

Arguments

- <list>

  (required) Specifies the Tcl list to target with **lsublist**.

- <pattern>

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

Examples

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

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

Related Topics

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

mem compare

This command compares a selected memory to a reference memory or file. Must have the "diff" utility installed and visible in your search path in order to run this command.

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

Syntax

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

Arguments

- -mem <ref_mem>

  (optional) Specifies a reference memory to be compared with actual_mem.

  <ref_mem> — A memory record.

- -file <ref_file>

  (optional) Specifies a reference file to be compared with actual_mem.

  <ref_file> — A saved memory file.

- actual_mem

  (required) Specifies the name of the memory to be compared against the reference data. Must be specified as the final argument to the **mem compare** command.

mem display

This command prints to the Transcript window the memory contents of the specified instance. If the given instance path contains only a single array signal or variable, the signal or variable name need not be specified.

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

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

By default, identical data lines are printed. To replace identical lines with a single line containing the asterisk character, you can enable compression with the **-compress** argument.

Note

The format settings are stored at the top of this file as a pseudo comment so that subsequent mem load commands can correctly interpret the data. Do not edit this data when manipulating a saved file.

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

Syntax

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

Arguments

- -addressradix [d | h]

  (optional) Specifies the address radix for the default (MTI) formatted files.

  d — Decimal radix. (default if -format is specified as mti.)

  h — Hex radix.

- -compress

  (optional) Specifies that identical lines not be printed. Reduces the file size by replacing exact matches with a single line containing an asterisk. These compressed files are automatically expanded during a **mem load** operation.

- -dataradix <radix_type>

  (optional) Specifies the data radix for the default (MTI) formatted files. If unspecified, the global default radix is used.

  <radix_type> A specified radix type. Valid entries (or any unique abbreviations) are: binary, decimal, unsigned, octal, hex, symbolic, and default. If no radix is specified for an enumerated type, the symbolic representation is used.

You can change the default radix type for the current simulation using the radix command or make the default radix permanent by editing the DefaultRadix variable in the modelsim.ini file.

- -endaddress <end>

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

  <end> — Any valid address in the memory. If unspecified, the default is the end of the memory.

- -format [bin | hex | mti]

  (optional) Specifies the output format of the contents.

  bin— Specifies a binary output.

  hex— Specifies a hex output.

  mti — MTI format. (default).

- -noaddress

  (optional) Specifies that addresses not be printed.

- -startaddress <st>

  (optional) Specifies the start address for a range of addresses to be displayed.

  <st> — Any valid address in the memory. If unspecified, the default is the start of the memory.

- -wordsperline <n>

  (optional) Specifies how many words are to be printed on each line.

  <n> — Any positive integer where the default is an 80 column display width.

- <path>

  (required) Specifies the full path to the memory instance. The default is the current context, as shown in the Structure window. Indexes can be specified. Must be specified as the final argument to the **mem display** command.

Examples

- This command displays the memory contents of instance */top/c/mru_mem*, addresses 5 to 10:

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

  returns:

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

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

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

  returns:

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

Related Topics

- For details on MTI format, refer to the mem load description.

mem list

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

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

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

Syntax

mem list [-r] [<path>]

Arguments

- -r

  (optional) Recursively descends into sub-modules when listing memories.

- <path>

  (optional) The hierarchical path to the location the search should start where the default is the current context, as shown in the Structure window.

Examples

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

  mem list -r /

  Returns:

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

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

  mem list /top2/uut -r

  Returns:

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

mem load

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

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

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

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

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

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

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

Syntax

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

Arguments

- -infile <infile>

  (required unless the **-filldata** argument is used) Updates memory data from the specified file.

  <infile> — The name of a memory file.

- -endaddress <end>

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

  <end> — Specified as any valid address in the memory.

- -filltype {dec | inc | rand | value}

  (optional, use with the **-filldata** argument) Fills in memory addresses in an algorithmic pattern starting with the data word specified in -filldata. If a fill pattern is used without a file option, the entire memory or specified address range is filled with the specified pattern. If both are specified, the pattern is applied only to memory locations not contained in the file.

  dec — Decrement each succeeding memory word by one digit.

  inc — Increment each succeeding memory word by one digit.

  rand — Randomly generate each succeeding memory word starting with the word specified by -filldata as the seed.

  value — Value (default) Substitute each memory word in the range with the value specified in -filldata.

- -filldata <data_word>

  (required unless **-infile** is used) Specifies a data word used to fill memory addresses in the pattern specified by -filltype.

  <data_word> — Specifies a data word. Must be in the same format as specified by the -fillradix switch.

- -fillradix <radix_type>

  (optional, use with **-filldata**) Specifies radix of the data specified by the -filldata switch.

  <radix_type> — Valid entries (or any unique abbreviations) are: binary, decimal, unsigned, octal, hex, symbolic, and default.

- -format [bin | hex | mti]

  (optional, use with **-infile**) Specifies the format of the file to be loaded.

  bin — Specifies binary data format.

  hex — Specifies hex format.

  mti — MTI format. (default).

  Specifies the format of the file to be loaded. The bin and hex values are the standard Verilog hex and binary memory pattern file formats. These can be used with Verilog memories, and with VHDL memories composed of std_logic types.

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

- <path>

  (optional) The hierarchical path to the memory instance. If the memory instance name is unique, shorthand instance names can be used. The default is the current context, as shown in the Structure window.

  Memory address indexes can be specified in the instance name also. If addresses are specified both in the instance name and the file, only the intersection of the two address ranges is populated with memory data.

- -skip <Nwords>

  (optional) Specifies the number of words to be skipped between each fill pattern value. Used with **-filltype** and **-filldata**.

  <Nwords> — Specified as an unsigned integer.

- -startaddress <st>

  (optional) Specifies the start address for a range of addresses to be loaded.

  <st> — Any valid address in the memory.

- -truncate

  (optional) Ignores any most significant bits (msb) in a memory word which exceed the memory word size. By default, when memory word size is exceeded, an error results.

Examples

- Load the memory pattern from the file *vals.mem* to the memory instance */top/m/mem*, filling the rest of the memory with the fixed-value 1‘b0.

  mem load -infile vals.mem -format bin -filltype value -filldata 1'b0 /top/m/mem

  When you enter the **mem display** command on memory addresses 0 through 12, you see the following:

```
mem display -startaddress 0 -endaddress 12 /top/m/mem
#  0: 0000000000000000 0000000000000001 0000000000000010 0000000000000011
#  4: 0000000000000100 0000000000000101 0000000000000110 0000000000000111
#  8: 0000000000001000 0000000000001001 0000000000000000 0000000000000000
# 12: 0000000000000000
```

- Load the memory pattern from the file *vals.mem* to the memory instance */top/m/mru_mem*, filling the rest of the memory with the fixed-value 16'Hbeef.

  mem load -infile vals.mem -format hex -st 0 -end 12 -filltype value -filldata 16'Hbeef /top/m/mru_mem

- Load memory instance */top/mem2* with two words of memory data using the Verilog Hex format, skipping 3 words after each fill pattern sequence.

  mem load -filltype value -filldata "16'hab 16'hcd" /top/mem2 -skip 3

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

  mem load -filldata 0 /top/mem

- Truncate the msb bits that exceed the maximum word size (specified in HDL code).

  mem load -format h -truncate -infile data_files/data.out /top/m_reg_inc/mem

Related Topics

- mem save

mem save

This command saves the contents of a memory instance to a file in any of the supported formats: Verilog binary, Verilog hex, and MTI memory pattern data.

This command works identically to the **mem display** command, except that its output is written to a file rather than a display.

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

You can use the **mem save** command to generate relocatable memory data files. The **-noaddress** option omits the address information from the memory data file. You can later load the generated memory data file using the **memory load** command.

Syntax

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

Arguments

- -outfile <filename>

  (required) Specifies that the memory contents are to be stored in a file.

  <filename> — The name of the file where the specified memory contents are to be stored.

- -addressradix {dec | hex}

  (optional) Specifies the address radix for the default mti formatted files.

  dec — Decimal (default).

  hex — Hexadecimal.

- -compress

  (optional) Specifies that only unique lines are printed, identical lines are not printed. Mutually exclusive with the **-noaddress** switch.

- -dataradix <radix_type>

  (optional) Specifies the data radix for the default mti formatted files.

  <radix_type> — Valid entries (or any unique abbreviations) are: binary, decimal, unsigned, octal, hex, and symbolic.

  You can change the default radix for the current simulation using the radix command. You can change the default radix permanently by editing the DefaultRadix variable in the modelsim.ini file.

- -endaddress <end>

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

  <end> — Any valid address in the memory.

- -format {bin | hex | mti}

  (optional) Specifies the format of the output file.

  bin— Binary data format.

  hex— Hexadecimal format.

  mti — MTI format. (default).

  The bin and hex values are the standard Verilog hex and binary memory pattern file formats. These can be used with Verilog memories, and with VHDL memories composed of std_logic types.

  In the MTI memory data file format, internal file address and data radix settings are stored within the file itself.

- -noaddress

  (optional) Prevents addresses from being printed. Mutually exclusive with the **-compress** switch.

- <path>

  (optional) The hierarchical path to the location of the memory instance. The default is the current context, as shown in the Structure window.

- -startaddress <st>

  (optional) Specifies the start address for a range of addresses to be saved.

  <st> — Any valid address in the memory.

- -wordsperline <Nwords>

  (optional) Specifies how many memory values are to be printed on each line.

  <Nwords> — Any unsigned integer where the default assumes an 80 character display width.

Examples

- Save the memory contents of the instance */top/m/mem(0:10)* to *memfile*, written in the mti radix.

  mem save -format mti -outfile memfile -start 0 -end 10 /top/m/mem

  The contents of *memfile* are as follows:

```
// memory data file (do not edit the following line - required for mem
load use)
// format=mti addressradix=d dataradix=s version = 1.0
0: 0000000000000000 0000000000000001 0000000000000010 0000000000000011
4: 0000000000000100 0000000000000101 0000000000000110 0000000000000111
8: 0000000000001000 0000000000001001 xxxxxxxxxxxxxxxx
```

Related Topics

- mem display
- mem load

mem search

This command finds and prints to the screen the first occurring match of a specified memory pattern in the specified memory instance. Shorthand instance names are accepted.

Optionally, you can instruct the command to print all occurrences. The search pattern can be one word or a sequence of words.

Syntax

mem search {**-glob <word> [<word>…]** | **-regexp <word> [<word>…]**}
[-addressradix {dec | hex}] [-dataradix <radix_type>] [-all] [-replace <word> [<word>…]]
[-startaddress <address>] [-endaddress <address>] [<path>]

Arguments

- -glob <word> [<word>…]

  (required unless using **-regexp**) Specifies the value of the pattern, accepting glob pattern syntax for the search.

  <word> — Any word pattern. Multiple word patterns are specified as a space separated list. Wildcards are accepted in the pattern.

  This argument and **-regexp** are mutually exclusive arguments.

- -regexp <word> [<word>…]

  (required unless using -glob) Specifies the value of the pattern, accepting regular expression syntax for the search.

  <word> — Any word pattern. Wildcards are accepted in the pattern. Multiple word patterns are specified as a space separated list.

  This argument and **-glob** are mutually exclusive arguments.

- -addressradix {dec | hex}

  (optional) Specifies the radix for the address being displayed.

  dec — Decimal (default).

  hex — Hexadecimal.

- -all

  (optional) Searches the specified memory range and returns all matching occurrences to the transcript. By default only the first matching occurrence is printed.

- -dataradix <radix_type>

  (optional) Specifies the radix for the memory data being displayed.

  <radix_type> — Can be specified as symbolic, binary, octal, decimal, unsigned, or hex. By default the radix displayed is the system default.

You can change the default radix for the current simulation using the radix command. You can change the default radix permanently by editing the DefaultRadix variable in the modelsim.ini file.

- -endaddress <address>

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

  <address> — Any valid address in the memory.

- <path>

  (optional) Specifies the hierarchical path to the location of the memory instance. The default is the current context, as shown in the Structure window.

- -replace <word> [<word>…]

  (optional) Replaces the found patterns with a designated pattern.

  <word> — A word pattern Multiple word patterns are accepted, separated by a single space. No wildcards are allowed in the replaced pattern.

- -startaddress <address>

  (optional) Specifies the start address for a range of addresses to search.

  <address> — Any valid address in the memory.

Examples

- Search for and print to the screen all occurrences of the pattern **16‘Hbeef** in */uut/u0/mem3*:

  mem search -glob 16‘Hbeef -dataradix hex /uut/u0/mem3

  Returns:

  ```
  #7845: beef
  #7846: beef
  #100223: beef
  ```

- Search for and print only the first occurrence of **16‘Hbeef** in the address range 7845:150000, replacing it with **16‘Hcafe** in */uut/u1/mem3*:

  mem search -glob 16‘Hbeef -d hex -replace 16‘Hcafe -st 7846 -end 150000 /uut/u1/mem3

  Returns:

  ```
  #7846: cafe
  ```

- Replace all occurrences of **16‘Hbeef** with **16‘Habe** in */uut/u1/mem3*:

  mem search -glob 16‘Hbeef -r 16‘Habe -addressadix hex -all /uut/u1/mem3

  Returns:

```
#1ea5: 2750
#1ea6: 2750
#1877f: 2750
```

- Search for and print the first occurrence any pattern ending in f:

  mem search -glob "*f"

- Search for and print the first occurrence of this multiple word pattern:

  mem search -glob "abe cafe" /uut/u1/mem3

- Search for patterns "0000 0000" or "0001 0000" in *m/mem*:

  mem search -data hex -regexp {000[0|1] 0{4}} m/mem -all

- Search for a pattern that has any number of 0s followed by any number of 1s as a memory location, and which has a memory location containing digits as the value:

  mem search -regexp {^0+1+$ \d+} m/mem -all

- Search for any initialized location in a VHDL memory:

  mem search -regexp {[^U]} -all <vhdl_memory>

messages clearfilter

This command removes any filter you have set in the Message Viewer. Refer to the section "Message Viewer Filter Dialog Box" for additional information about filtering in the Message Viewer.

Syntax

messages clearfilter

Arguments

- No arguments

messages setfilter

This command performs the same action as the Message Viewer Filter Dialog Box, which controls which messages are shown in the Message Viewer.

The ideal workflow for using this command is through the GUI:

1. **View > Message Viewer**.
2. Right-click in the Message Viewer and select **Filter**.

   The Message Viewer Filter dialog box is displayed
3. Create your filter.
4. **OK** or **Apply**.

   The Message Viewer updates based on your filter and a messages setfilter command, which is equivalent to your settings, is output to the transcript.
5. Retain the messages setfilter command from the transcript for future use.

Syntax

messages setfilter <tcl_list>

Arguments

- <tcl_list> — The tcl_list argument is a complex string of tcl code that controls the filter settings.

Examples

- Severity is error and time is greater than or equal to 100 ns

```
messages setfilter {{} \
   ( Severity Contains {Case Insensitive} error )} \
   {AND ( Time >= 100 ns )}
```

- The objects field contains neither clock or reset

```
messages setfilter {{} \
   ( Object Contains {Case Sensitive} clock )} \
   {NOR ( Object Contains {Case Sensitive} data )}
```

- The message string either contains reg_str2 or reg_str1

```
messages setfilter {{} \
   ( Message Contains {Case Insensitive} reg_str2 )} \
   {OR ( Message Contains {Case Insensitive} reg_str1 )}
```

messages write

This command prints the contents of the Message Viewer window to a specified text file.

Syntax

messages write **<filename>**

Arguments

- <filename> — (required) Specifies the name of the file where the contents of the Message Viewer window are to be saved.

Related Topics

- displaymsgmode modeslim.ini variable
- msgmode modeslim.ini variable
- "Message Viewer Window"

modelsim

The **modelsim** command starts the ModelSim GUI without prompting you to load a design.

This command is valid only for Windows platforms and may be invoked in one of three ways:

- from the DOS prompt
- from a ModelSim shortcut
- from the Windows Start > Run menu

To use **modelsim** arguments with a shortcut, add them to the target line of the properties of that shortcut. (As expected, arguments also work on the DOS command line.)

You can invoke the simulator from either the ModelSim> prompt after the GUI starts or from a DO file called by **modelsim**.

Syntax

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

Arguments

- -do <macrofile>

  (optional) Executes a DO file when **modelsim** is invoked.

  <macrofile> — The name of a DO file

Note

In addition to the macro called by this argument, if a DO file is specified by the STARTUP variable in *modelsim.ini*, it will be called when the vsim command is invoked.

- <license_option>

  (optional) Restricts the search of the license manager.

- -nosplash

  (optional) Disables the splash screen.

Related Topics

- do
- vsim
- Using a Startup File

noforce

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

Syntax

noforce **<object_name> …**

Arguments

- <object_name>

  (required) Specifies the name of an object. Must match an object name used in a previous force command. Multiple object names may be specified as a space separated list. Wildcard characters are allowed.

Related Topics

- force
- Wildcard Characters

nolog

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

A flag is written into the WLF file for each signal turned off, and the GUI displays "-No Data-" for the signal(s) until logging (for the signal(s)) is turned back on. Logging can be turned back on by issuing another log command or by doing a **nolog -reset**.

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

Transactions written in SCV or Verilog are logged automatically, and can be removed with the **nolog** command. A transaction is logged into the *.wlf* file if logging is enabled (in other words, if no **nolog** command has disabled it) for that stream at the time when the transaction was begun. An entire span of a transaction is either logged or not logged, regardless of the begin and end times specified for that transaction.

Syntax

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

Arguments

- -all

  (optional) Turns off logging for all signals currently logged.

- -depth <level>

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

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

- -howmany

  (optional) Returns an integer indicating the number of signals found.

- -in

  (optional) Turns off logging only for ports of mode IN whose names match the specification.

- -inout

  (optional) Turns off logging only for ports of mode INOUT whose names match the specification.

- -internal

  (optional) Turns off logging only for internal (non-port) objects whose names match the specification.

- -out

  (optional) Turns off logging only for ports of mode OUT whose names match the specification.

- -ports

  (optional) Specifies that the scope of the search is to include all ports.

- -recursive

  (optional) Specifies that the scope of the search is to descend recursively into subregions. If omitted, the search is limited to the selected region. You can use the **-depth** argument to specify how many levels of the hierarchy to descend.

- -reset

  (optional) Turns logging back on for all unlogged signals.

- <object_name>…

  (optional) Specifies the object name which you want to unlog. Multiple object names may be specified as a space separated list. Wildcard characters are allowed.

Examples

- Unlog all objects in the design.

  ```
  nolog -r /*
  ```

- Turn logging back on for all unlogged signals.

  ```
  nolog -reset
  ```

Related Topics

- add list
- add wave
- log

notepad

This command opens a simple text editor. It may be used to view and edit ASCII files or create new files.

This mode can be changed from the Notepad Edit menu.

Returns nothing.

Syntax

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

Arguments

- <filename>

  (optional) Name of the file to be displayed.

- -r

  (optional) Specifies read-only mode.

- -edit

  (optional) Specifies editing mode. Will not save changes to an existing file that has the Read-only attribute turned on. (default)

noview

This command closes a window in the ModelSim GUI. To open a window, use the **view** command.

Syntax

noview **<window_name>**…

Arguments

- <window_name>…

  (required) Specifies the window(s) to close. Multiple window types may be specified in a space separated list. Wildcards permitted. At least one type (or wildcard) is required.

  Refer to the view command for a complete list of possible arguments.

  You can also close Source windows using the tab or file name.

Examples

- Close the Wave window named "wave1".

  noview wave1

- Close all List windows.

  noview List

Related Topics

- view

nowhen

This command deactivates selected when commands.

Syntax

nowhen [<label>]

Arguments

- <label>

  (optional) Specifies an individual when command. Wildcards may be used to select more than one when command.

Examples

- Deactivate the when command labeled 99.

  nowhen 99

- Deactivate all when commands.

  nowhen *

Related Topics

- when

onbreak

This command is used within a macro and specifies one or more scripts to be executed when running a macro that encounters a breakpoint in the source code.

The onbreak setting will effect any run commands that follow the onbreak statement until another onbreak command is issued. If a *.do* file is executed from within the macro, the *.do* file script will inherit the onbreak setting specified prior to execution unless and until another onbreak command is given, in which case, that onbreak setting will be in effect until the *.do* file script completes at which point execution will return to the calling macro and the calling macro's onbreak setting is restored. The script must be followed by a run command to take effect.

Use an empty string to change the **onbreak** command back to its default behavior:

```
onbreak ""
```

In this case, the macro will be interrupted after a breakpoint occurs (after any associated bp command string is executed).

If you specify this command in a macro without a script, the default behavior is to pause and return control to the command line. You can then enter onbreak without arguments on the command line after the macro has paused to return the current onbreak command string.

Syntax

onbreak <script>; <script>...

Arguments

- <script>

  (optional) Any command or script can be used as an argument to **onbreak**. If you want to use more than one command or script, use a semicolon to separate them or place them on multiple lines and enclose the entire script in curly braces ({}) or quotation marks (“ “). You must use the **onbreak** command before a **run**, **run -continue**, or **step** command. If a **run** or **step** comand is issued within an onbreak script, the script will return immediately and any following commands will not be executed. It is an error to execute any commands within an **onbreak** command string following any of the **run** commands. This restriction applies to any macros or Tcl procedures used in the **onbreak** command string.

Examples

- Examine the value of the HDL object data when a breakpoint is encountered. Then continue the run command.

  ```
  onbreak {exa data ; cont}
  ```

- Resume execution of the macro file on encountering a breakpoint.

  ```
  onbreak resume
  ```

- This set of commands test for assertions. Assertions are treated as breakpoints if the severity level is greater than or equal to the current BreakOnAssertion variable setting (refer to modelsim.ini Variables). By default a severity level of failure or above causes a breakpoint; a severity level of error or below does not.

```
set broken 0
onbreak {
  lassign [runStatus -full] status fullstat
  if {$status eq "error"} {
    # Unexpected error, report info and force an error exit
    echo "Error: $fullstat"
    set broken 1
    resume
  } elseif {$status eq "break"} {
    # If this is a user break, then
    # issue a prompt to give interactive
    # control to the user
    if {[string match "user_*" $fullstat]} {
      pause
    } else {
      # Assertion or other break condition
      set broken 2
      resume
    }
  } else {
    resume
  }
}
run -all
if {$broken == 1} {
  # Unexpected condition.  Exit with bad status.
  echo "failure"
  quit -force -code 3
} elseif {$broken == 2} {
  # Assertion or other break condition
  echo "error"
  quit -force -code 1
} else {
  echo "success!"
}
quit -force
```

Related Topics

- abort
- bp
- do
- onerror
- resume
- status
- Useful Commands for Handling Breakpoints and Errors
- Macros (DO Files)

onElabError

This command specifies one or more commands to be executed when an error is encountered during the elaboration portion of a **vsim** command. The command is used by placing it within a macro.

Use the **onElabError** command without arguments to return to a prompt.

Syntax

onElabError {[<command> [; <command>] …]}

Arguments

- <command>

  (optional) Any command can be used as an argument to **onElabError**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces ({}).

Related Topics

- do

onerror

This command is used within a macro before a **run** command; it specifies one or more commands to be executed when a running macro encounters an error.

Using the **onerror** command without arguments will return the current **onerror** command string. Use an empty string (**onerror** ““) to change the **onerror** command back to its default behavior. Use **onerror** with a **resume** command to allow an error message to be printed without halting the execution of the macro file.

You can also set the global OnErrorDefaultAction Tcl variable to dictate what action ModelSim takes when an error occurs. To set the variable on a permanent basis, you must define the variable in a *modelsim.tcl* file (Refer to “The modelsim.tcl File” for details).

When your **onerror** command is successful, the macro will continue normally, unless your command instructs the tool to quit, for example:

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

or

```
onerror {break}
```

However, if your **onerror** command is not successful, the simulator will be halted, for example:

```
onerrror {add wave b}
```

when you don’t have a signal named b.

The **onerror** command is executed when a Tcl command (for example, **break**.) encounters an error in the macro file that contains the **onerror** command (note that a **run** command does not necessarily need to be in process). Conversely, OnErrorDefaultAction will run even if the macro does not contain a local **onerror** command. This can be useful when you run a series of macros from one script, and you want the same behavior across all macros.

Syntax

onerror {[<command> [; <command>] …]}

Arguments

- <command>

  (optional) Any command can be used as an argument to **onerror**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces ({}).

Example

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

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

Related Topics

- abort
- do
- onbreak
- resume
- status
- Useful Commands for Handling Breakpoints and Errors
- Macros (DO Files)

onfinish

This command controls simulator behavior when encountering $finish or sc_stop() in the design code. When you specify this command without an argument, it returns the current setting.

Syntax

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

Arguments

ask — (optional) In batch mode, the simulation will exit; in GUI mode, the user is prompted for action.

exit — (optional) The simulation exits without asking for any confirmation.

final — (optional) The simulation executes all finish blocks before exiting.

stop — (optional) The simulation ends but remains loaded in memory, allowing for easier post-simulation tasks.

default — (optional) Uses the current setting for the OnFinish variable in the *modelsim.ini* file.

Related Topics

- OnFinish modelsim.ini variable
- vsim -onfinish

pause

This command interrupts the execution of a macro and allows you to perform interactive debugging of a macro file. The command is placed within the macro to be debugged.

Syntax

pause

Arguments

- None.

Description

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

```
VSIM(paused)>
```

This “pause” prompt notifies you that a macro has been interrupted.

When a macro is paused, you can invoke another macro. If the second macro is interrupted, you can continue invoking macros up to a nesting level of 50 macros.

The status command lists summary information about all interrupted macros.

Use the resume command to resume execution of the macro. Use the abort command to stop execution of some or all of the macros.

Related Topics

- abort
- do
- resume
- run
- status

precision

This command determines how real numbers display in the graphic interface (e.g., Objects, Wave, Locals, and List windows). It does not affect the internal representation of a real number and therefore precision values over 17 are not allowed.

Executing the **precision** command without any arguments returns the current precision setting.

Syntax

precision [<digits>[#]]

Arguments

- <digits>[#]

  (optional) Specifies the number of digits to display where the default is 6.

  # — A suffix that forces the display of trailing zeros. See examples for more details.

Examples

- Results in 4 digits of precision.

  precision 4

  For example:

  ```
  1.234 or 6543
  ```

- Results in 8 digits of precision including trailing zeros.

  precision 8#

  For example:

  ```
  1.2345600 or 6543.2100
  ```

- Results in 8 digits of precision but doesn’t print trailing zeros.

  precision 8

  For example:

  ```
  1.23456 or 6543.21
  ```

printenv

This command prints to the Transcript window the current names and values of all environment variables.

If variable names are given as arguments, returns only the names and values of the specified variables.

Syntax

printenv [<var>…]

Arguments

- <var>…

  (optional) Specifies the name(s) of the environment variable(s) to print.

Examples

- Print all environment variable names and their current values.

  printenv

  Returns:

  ```
  # CC = gcc
  # DISPLAY = srl:0.0
  …
  ```

- Print the specified environment variables:

  printenv USER HOME

  Returns:

  ```
  # USER = vince
  # HOME = /scratch/srl/vince
  ```

process report

This command creates a textual report of all processes displayed in the Process Window.

Syntax

process report [-file <filename>] [-append]

Arguments

- -file <filename>

  (optional) Creates an external file where raw process data will be saved. If **-file** is not specified, then the output is redirected to stdout.

  <filename> — A user-specified name for the file.

- -append

  (optional) Specifies that process data is to be appended to the current process report file. If this option is not used, the process data will overwrite the existing process report file.

project

This command is used to perform common operations on projects.

Prerequisites

Some arguments to this command require a project to be opened with either the **project new** or **project open** command. Some argument must be used outside of a simulation session. Please read the argument descriptions for more information.

Syntax

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

Arguments

- addfile **<filename>** [<file_type>] [<folder_name>]

  (optional) Adds the specified file to the current project. Requires a project to be open.

  <filename> — (required) The name of an existing file.

  <file_type> — (optional) The HDL file type of the file being added. For example do for a *.do* file.

  <folder_name> — (optional) Places the file in an existing folder created with **project addfolder** command. If no folder name is specified the file will be placed in the top level folder.

- addfolder <foldername> [<folder_parent>]

  (optional) Creates a project folder within the project. Requires a project to be open.

  <foldername> — (required) Any string.

  <folder_parent> — (optional) Places <foldername> in an existing parent folder. If <folder_parent> is unspecified, <foldername> is placed at the top level.

- calculateorder

  (optional) Determines the compile order for the project by compiling each file, then moving any compiles that fail to the end of the list. This is repeated until there are no more compile errors.

- close

  (optional) Closes the current project.

- compileall [-n]

  (optional) Compiles all files in the project using the defined compile order.

  -n — (optional) Returns a list of the compile commands this command would execute, without actually executing the compiles.

- compileorder

  (optional) Returns the current compile order list.

- compileoutofdate [-n]

  (optional) Compiles all files that have a newer date/time stamp than the last time the file was compiled.

  -n — Returns a list of the compile commands this command would execute, without actually executing the compiles.

- delete <filename>

  (optional) Deletes a project file.

  <filename> — Any *.mpf* file.

- filenames

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

- env

  (optional) Returns the current project file and path.

- history

  (optional) Lists a history of manipulated projects. Must be used outside of a simulation session.

- new <home_dir> <proj_name> [<defaultlibrary>] [<intialini>] [0 | 1]

  (optional) Creates a new project under a specified home directory with a specified name and optionally a default library. The name of the work library will default to "work" unless specified. A new project cannot be created while a project is currently open or a simulation is in progress.

  <home_dir> — The path to the new project directory within the current working directory.

  <proj_name> — Specifies a name for the new project. The file will be saved as an *.mpf* file

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

  <intialini> — An optional *modelsim.ini* file can be specified as a seed for the project file by using the initialini option. If initialini is an empty string, then ModelSim uses the current *modelsim.ini* file when creating the project. You must specify a default library if you want to specify initialini.

  0 — (default) Copies all library mappings from the specified <initialini> file into the new project.

  1 — Copies library mappings referenced in an "others" clause in the initial *.ini* file.

- open <project>

  (optional) Closes any currently opened project and opens a specified project file (must be a valid *.mpf* file), making it the current project. Changes the current working directory to the project's directory. Must be used outside of a simulation session.

- removefile <filename>

  (optional) Removes the specified file from the current project.

Examples

- Make */user/george/design/test3/test3.mpf* the current project and changes the current working directory to */user/george/design/test3*.

  project open /user/george/design/test3/test3.mpf

- Execute current project library build scripts.

  project compileall

pwd

This Tcl command displays the current directory path in the Transcript window.

Syntax

pwd

Arguments

- None

quietly

This command turns off transcript echoing for the specified command.

Syntax

quietly **<command>**

Arguments

- <command>

  (required) Specifies the command for which to disable transcript echoing. Any results normally echoed by the specified command will not be written to the Transcript window. To disable echoing for all commands use the transcript command with the **-quietly** option.

Related Topics

- transcript

quit

This command exits the simulator.

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

Syntax

quit [-f | -force] [-sim] [-code <integer>]

Arguments

- -f | -force

  (optional) Quits without asking for confirmation. If omitted, ModelSim asks you for confirmation before exiting. (The -f and -force arguments are equivalent.)

- -sim

  (optional) Unloads the current design in the simulator without exiting ModelSim. All files opened by the simulation will be closed including the WLF file (*vsim.wlf*).

- -code <integer>

  (optional) Quits the simulation and issues an exit code.

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

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

Examples

Refer to the Examples section of the exit command for an example of using the **-code** argument. The quit and exit commands behave similarly in this regard.

radix

This command specifies the default radix to be used for the current simulation. Specifying the command with no argument returns the current radix setting.

The command can be used at any time. The specified radix is used for all commands (force, examine, change, etc.) as well as for displayed values in the Objects, Locals, Dataflow, List, and Wave windows, as well as the Source window in the source annotation view.

Alternate methods for changing the default radix:

- In the *modelsim.ini* file, edit the DefaultRadix variable.
- Choose **Simulate > Runtime Options** from the main menu, click the **Defaults** tab, make your selection in the **Default Radix** box.

Syntax

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

Arguments

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

- -ascii

  (optional) Display a Verilog object as a string equivalent using 8-bit character encoding.
- -binary

  (optional) Displays values in binary format.
- -enumnumeric

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

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

  (optional) Displays values in fixed-point format.
- -decimal

  (optional) Displays values in decimal format. You can specify **-signed** as an alias for this argument.
- -hexadecimal

  (optional) Displays values in hexadecimal format.

- -octal

  (optional) Displays values in octal format.

- -time

  (optional) Displays values of time for register-based types in Verilog.

- -showbase

  (optional) Display the number of bits of the vector and the radix used, where:

  binary = b
  decimal = d
  hexidecimal = h
  ASCII = a
  time = t

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

- -symbolic

  (optional) Displays values in a form closest to their natural form.

- -unsigned

  (optional) Displays values in unsigned decimal format.

Related Topics

- User-Defined Radices
- radix define
- radix delete
- radix names
- radix list
- radix signal

radix define

This command is used to create or modify a user-defined radix. A user definable radix is used to map bit patterns to a set of enumeration labels. User-defined radices are available for use in the Wave and List windows or with the examine command.

Syntax

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

Arguments

- <name>

  (required) User-specified name for the radix.

- <definition_body>

  (required) A list of number pattern, label pairs. The definition body has the form:

```
{
   <numeric-value> <enum-label>,
   <numeric-value> <enum-label>
   -default <radix_type>
}
```

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

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

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

  The comma (,) in the definition body is optional. The <enum-label> is any arbitrary string. It should surrounded by quotation marks (""), especially if it contains spaces.

  The **-default** entry is optional. If present, it defines the radix to use if a match is not found for a given value. The **-default** entry can appear anywhere in the list, it does not have to be at the end.

- -color <value>

  (optional) Designates a color for the waveform and text in the Wave window.

  <value> — The color value may be a color name or its hex value (see example below).

Example

- The radix define command used to create a radix called “States,” which will display state values in the List, Watch, and Wave windows instead of numeric values.

```
radix define States {
    11'b00000000001 "IDLE",
```

```
    11'b00000000010 "CTRL",
    11'b00000000100 "WT_WD_1",
     11'b00000001000 "WT_WD_2",
     11'b00000010000 "WT_BLK_1",
     11'b00000100000 "WT_BLK_2",
    11'b00001000000 "WT_BLK_3",
    11'b00010000000 "WT_BLK_4",
    11'b00100000000 "WT_BLK_5",
    11'b01000000000 "RD_WD_1",
    11'b10000000000 "RD_WD_2",
    -default hex
}
```

- The following example illustrates how to specify the radix color:

```
radix define States {
    11'b00000000001 "IDLE" -color yellow,
    11'b00000000010 "CTRL" -color #ffee00,
    11'b00000000100 "WT_WD_1" -color orange,
    11'b00000001000 "WT_WD_2" -color orange,
    11'b00000010000 "WT_BLK_1",
    11'b00000100000 "WT_BLK_2",
    11'b00001000000 "WT_BLK_3",
    11'b00010000000 "WT_BLK_4",
    11'b00100000000 "WT_BLK_5",
    11'b01000000000 "RD_WD_1" -color green,
    11'b10000000000 "RD_WD_2" -color green,
    -default hex
    -defaultcolor white
}
```

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

To specify a range of values, wildcards may be specified for bits or characters of the value. The wildcard character is '?', similar to the iteration character in a Verilog UDP, for example:

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

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

Related Topics

- User-Defined Radices
- radix
- radix delete
- radix names
- radix list
- radix signal

radix delete

This command will remove the radix definition from the named radix.

Syntax

radix delete **<name>**

Arguments

- <name>

  (required) Removes the radix definition from the named radix.

Related Topics

- User-Defined Radices
- radix
- radix define
- radix list
- radix names
- radix signal

radix list

This command will return the complete definition of a radix, if a name is given. If no name is given, it will list all the defined radices.

Syntax

radix list [<name>]

Arguments

- <name>

  (optional) Returns the complete definition of the named radix.

Related Topics

- User-Defined Radices
- radix
- radix define
- radix delete
- radix names
- radix signal

radix names

This command returns a list of currently defined radix names.

Syntax

radix names

Arguments

None

Related Topics

- User-Defined Radices
- radix
- radix define
- radix delete
- radix list
- radix signal

radix signal

This command sets or inspects radix values for the specified signal in the Objects, Locals, Schematic, and Wave windows.

Note

The intent is for this command to be used for a small number of signals. If the majority of signals in a design are to use a particular radix value, then set that value as the default radix with the radix command, and use the **radix signal** command for the rest.

When no argument is used, the **radix signal** command returns a list of all signals with a radix.

Syntax

radix signal [<signal_name> [<radix_value>]] [-fpoint <decimal>] [-showbase]

Arguments

- <signal_name>

  (optional) Name of the signal for which the radix will be set (if <radix_value> is specified) or inspected.

- <radix_value>

  (optional) Value of the radix to be set for the specified signal. Use empty quotation marks ("") to unset the radix for the specified signal.

- -fpoint <decimal>

  (optional) Designates a fixed point radix with “decimal” specifying the number of decimal places of the radix.

- -showbase

  (optional) Display the number of bits of the vector and the radix used, where:

  binary = b
  decimal = d
  hexidecimal = h
  ASCII = a
  time = t

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

Related Topics

- User-Defined Radices
- radix
- radix define
- radix list
- radix delete

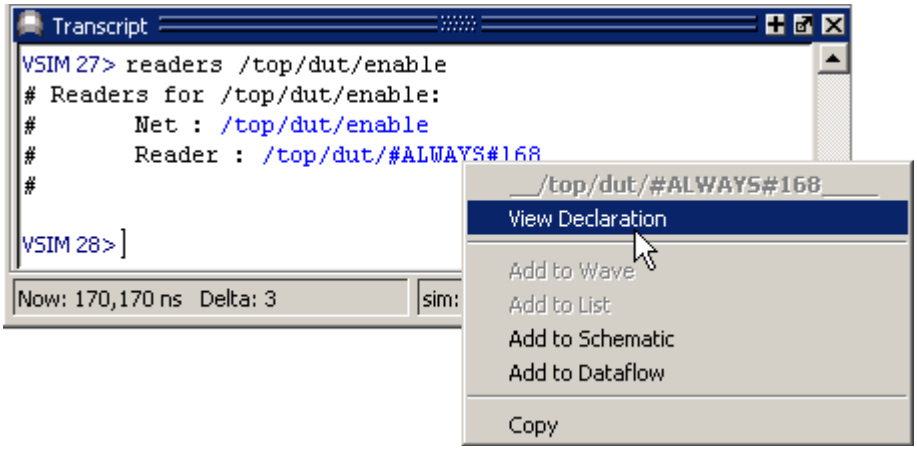
readers

This command displays the names of all readers of the specified object.

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

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

Figure 2-4. readers Command Results in Transcript



Syntax

readers **<object_name>** [-source]

Arguments

- <object_name>

  (required) Specifies the name of the signal or net whose readers are to be shown. All signal or net types are valid. Multiple names and wildcards are accepted.

- -source

  (optional) Returns the source file name and line number for each driver of the specified signal or net. If the source location cannot be determined, the value n/a is returned for that driver.

Related Topics

- drivers

report

This command displays information relevant to the current simulation.

Syntax

report files

report where [ini] [pwd] [transcript] [wlf] [project]

report simulator control

report simulator state

Arguments

- files

  Returns a list of all source files used in the loaded design. This information is also available in the Specified Path column of the Files window.

- where [ini] [pwd] [transcript] [wlf] [project]

  Returns a list of configuration files where the arguments limit the list to those files specified. If specified without arguments, returns a list of all configuration files in the current simulation.

  ini — (optional) Returns the location of the *modesim.ini* file.

  pwd — (optional) Returns the current working directory.

  transcript — (optional) Returns the location for saving the transcript file.

  wlf — (optional) Returns the current location for saving the *.wlf* file.

  project — (optional) Returns the current location of the project file.

- simulator control

  Displays the current values for all simulator control variables.

- simulator state

  Displays the simulator state variables relevant to the current simulation.

Examples

- Display configuration file information

  report where

  Returns:

  ```
  # INI {modelsim.ini}
  # PWD ./Testcases/
  # Transcript transcript
  # WLF vsim.wlf
  # Project {}
  ```

- Display all simulator control variables.

report simulator control

Returns:

```
# UserTimeUnit = ns
# RunLength =
# IterationLimit = 5000
# BreakOnAssertion = 3
# DefaultForceKind = default
# IgnoreNote = 0
# IgnoreWarning = 0
# IgnoreError = 0
# IgnoreFailure = 0
# IgnoreSVAInfo= 0
# IgnoreSVAWarning = 0
# IgnoreSVAError = 0
# IgnoreSVAFatal = 0
# CheckpointCompressMode = 1
# NumericStdNoWarnings = 0
# StdArithNoWarnings = 0
# PathSeparator = /
# DefaultRadix = symbolic
# DelayFileOpen = 1
# WLFFilename = vsim.wlf
# WLFTimeLimit = 0
# WLFSizeLimit = 0
```

- Display all simulator state variables. Only the variables that relate to the design being simulated are displayed:

  report simulator state

  Returns:

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

Viewing preference variables

Preference variables have more to do with the way things look (but not entirely) rather than controlling the simulator. You can view preference variables from the Preferences dialog box. Select **Tools > Edit Preferences** (Main window).

Related Topics

- modelsim.ini Variables
- Simulator GUI Preferences

restart

This command reloads the design elements and resets the simulation time to zero. Only design elements that have changed are reloaded. (Note that SDF files are always reread during a restart.)

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

Shared libraries are handled as follows during a restart:

- Shared libraries that implement VHDL foreign architectures only are reloaded at each restart when the architecture is elaborated (unless vsim **-keeploaded** is used).
- Shared libraries loaded from the command line (**-foreign** and **-pli** options) and from the Veriuser entry in the *modelsim.ini* file are reloaded (unless **vsim -keeploaded** is used).
- Shared libraries that implement VHDL foreign subprograms remain loaded (they are not reloaded) even if they also contain code for a foreign architecture.

You can configure defaults for the restart command by setting the **DefaultRestartOptions** variable in the *modelsim.ini* file. Refer to "Restart Command Defaults".

To handle restarts with Verilog PLI applications, you need to define a Verilog user-defined task or function, and register a misctf class of callback. To handle restarts with Verilog VPI applications, you need to register reset callbacks. To handle restarts with VHDL FLI applications, you need to register restart callbacks. Refer to "Verilog Interfaces to C" for more information on the Verilog PLI/VPI/DPI and the *ModelSim FLI Reference* for more information on the FLI.

Syntax

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

Arguments

- -force

  (optional) Specifies that the simulation will be restarted without requiring confirmation in a popup window.
- -nobreakpoint

  (optional) Specifies that all breakpoints will be removed when the simulation is restarted where the default is for all breakpoints to be reinstalled after the simulation is restarted.

- -nolist

  (optional) Specifies that the current List window environment will **not** be maintained after the simulation is restarted where the default is for all currently listed HDL objects and their formats to be maintained.

- -nolog

  (optional) Specifies that the current logging environment will **not** be maintained after the simulation is restarted where the default is for all currently logged objects to continue to be logged.

- -nowave

  (optional) Specifies that the current Wave window environment will **not** be maintained after the simulation is restarted where the default is for all objects displayed in the Wave window to remain in the window with the same format.

Related Topics

- vsim

resume

This command is used to resume execution of a macro (DO) file after a pause command or a breakpoint.

This command may be input manually or placed in an onbreak command string. (Placing a **resume** command in a bp command string does not have this effect.) The **resume** command can also be used in an onerror command string to allow an error message to be printed without halting the execution of the macro file.

Syntax

resume

Arguments

- None

Related Topics

- abort
- do
- onbreak
- onerror
- pause
- Useful Commands for Handling Breakpoints and Errors

run

This command advances the simulation by the specified number of timesteps.

Syntax

run [<timesteps>[<time_units>]] | -all | -continue | -init | -next | -over |
-step [-inst <full_path>] [-out] [-over] [-this "this==<class_handle>"]]

Arguments

- No arguments

  Runs the simulation for the default time (100 ns).

  You can change the default <timesteps> and <time_units> in the GUI with the Run Length toolbar box in the Simulate toolbar or from the *modelsim.ini* file: RunLength and UserTimeUnit variables.

- <timesteps>[<time_units>]

  (optional) Specifies the number of timesteps for the simulation to run. The number may be fractional, or may be specified as absolute by preceding the value with the character @.

  <time_units> — Any valid time unit: fs, ps, ns, us, ms, or sec where the default is to use the current time unit.

- -all

  (optional) Causes the simulator to run the current simulation forever, or until it hits a breakpoint or specified break event.

- -continue

  (optional) Continues the last simulation run after a run -step, run -step -over command or a breakpoint. A **run -continue** command may be input manually or used as the last command in a bp command string.

- -final

  (optional) Instructs the simulator to run all final blocks then exit the simulation.

- -init

  (optional) Initializes non-trivial static SystemVerilog variables before beginning the simulation, for example, expressions involving other variables and function calls,. This could be useful for when you want to initialize values before executing any force, examine, or bp commands.

  You cannot use **run -init** after any other run commands or if you have specified vsim **-runinit** on the command line because all variables would have been initialized by that point.

- -next

  (optional) Causes the simulator to run to the next event time.

- -step

  (optional) Steps the simulator to the next HDL . Current values of local HDL variables may be observed at this time using the Locals window. You can specify the following arguments when you use -step:

  -inst <full_path>

  (optional) Instructs the simulation to step into a specific instance, process, or thread.

  <full_path> — Specifies the full path to an instance, process or thread.

  -out

  (optional) Instructs the simulation to step out of the current function or procedure and return to the caller.

  -over

  (optional) Directs ModelSim to run VHDL procedures and functions, Verilog tasks and functions but to treat them as simple statements instead of entering and tracing them line by line.

  You can use the **-over** argument to skip over a VHDL procedures or functions, Verilog task or functions. When a wait statement or end of process is encountered, time advances to the next scheduled activity. ModelSim then updates the Process and Source windows to reflect the next activity.

  -this "this==<class_handle>"

  (optional) Instructs the simulation to step into a method of a SystemVerilog class when “this” refers to the specified class handle. To obtain the handle of the class, use the examine -handle command.

  <class_handle> — Specifies a SystemVerilog class. Note that you must use quotation marks (" ") with this argument.

- -over

  (optional) Directs ModelSim to run VHDL procedures and functions, Verilog tasks and functions but to treat them as simple statements instead of entering and tracing them line by line.

Examples

- Advance the simulator 1000 timesteps.

  ```
  run 1000
  ```

- Advance the simulator the appropriate number of timesteps corresponding to 10.4 milliseconds.

  ```
  run 10.4 ms
  ```

- Advance the simulator to timestep 8000.

  ```
  run @8000
  ```

- Advance the simulator into the instance /top/p.

```
run -step -inst /top/p
```

Related Topics

- Simulate Toolbar

runStatus

This command returns the current state of your simulation to stdout after issuing a run or step command.

Syntax

runStatus [-full]

Arguments

- -full

  (optional) Appends additional information to the output of the **runStatus** command.

Returns

Table 2-3 (runStatus Command States) and Table 2-4 (runStatus -full Command Information) show outputs of the runStatus command.

Table 2-3. runStatus Command States

State	Description
ready	The design is loaded and is ready to run.
break	The simulation stopped before completing the requested run.
error	The simulation stopped due to an error condition.
loading	The simulation is currently elaborating.
nodesign	There is no design loaded.
checkpoint	A checkpoint is being created, do not interrupt this process.
cready	The design is loaded and is ready to run in C debug mode.
initializing	The user interface initialization is in progress.

Table 2-4. runStatus -full Command Information

-full Information	Description
bkpt	stopped at breakpoint
bkpt_builtin	stopped at breakpoint on builtin process
end	reached end of requested run
fatal_error	encountered fatal error (such as, divide by 0)
iteration_limit	iteration limit reached, possible feedback loop
silent_halt	mti_BreakSilent() called,
step	run -step completed
step_builtin	run -step completed on builtin process

Table 2-4. runStatus -full Command Information (cont.)

-full Information	Description
step_wait_suspend	run -step completed, time advanced.
user_break	run interrupted do to break-key or ^C (SIGINT)
user_halt	mti_Break() called.
user_stop	stop or finish requested from vpi, stop command, etc.
gate_oscillation	Verilog gate iteration limit reached.
simulation_stop	pli stop_simulation() called.

searchlog

This command searches one or more of the currently open logfiles for a specified condition.

It can be used to search for rising or falling edges, for signals equal to a specified value, or for when a generalized expression becomes true.

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

Syntax

searchlog [-command <cmd>] [-count <n>] [-deltas] [-endtime <time> [<unit>]] [-env <path>] [-event <time>] [-expr {<expr>}] [-reverse] [-rising | -falling | -anyedge] [-startDelta <num>] [-value <string>] **<startTime> [<unit>] <pattern>**

Description

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

- -command <cmd>

  (optional) Specifies a Tcl command that will be called for each event on the specified signal.

  <cmd> — A Tcl command that receives four arguments and returns one of three values: "continue", "stop", or "" (empty).

  The command will be passed four arguments: the reason for the call, the time of the event, the delta for the event, and the value. The reason value will be one of WLF_STARTLOG, WLF_ENDLOG, WLF_EVENT, or WLF_WAKEUP. The function is expected to return "continue", "stop", or "" (empty). If "continue" or "" (empty) is returned, the search continues, making additional callbacks as necessary. If "stop" is returned, the search stops and control is returned to the caller of the searchlog command.

  Only searching for a single signal is supported.

- -count <n>

  (optional) Specifies to search for the nth occurrence of the match condition.

  <n> — Any positive integer.

- -deltas

  (optional) Indicates to test for a match on simulation delta cycles. Otherwise, matches are only tested for at the end of each simulation time step.

- -endtime <time> [<unit>]

  (optional) Specifies the simulation time at which to end the search. By default no end time is specified.

  <time> — Specified as an integer or decimal number. Current simulation units are the default unless specifying **<unit>**.

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

- -env <path>

  (optional) Indicates to test for a match on simulation delta cycles. Otherwise, matches are only tested for at the end of each simulation time step.ath> — A path to a design region. Wildcards not allowed.

- -event <time>

  (optional) Indicates to test for a match on a simulation event. Otherwise, matches are only tested for at the end of each simulation time step.

- -expr {<expr>}

  (optional) Specifies a search for a general expression of signal values and simulation time. **searchlog** will search until the expression evaluates to true.

  {<expr>} — An expression that evaluates to a boolean true. See GUI_expression_format for the format of the expression.

- -reverse

  (optional) Specifies to search backwards in time from **<startTime>**.

- -rising

  (optional) Specifies a search for rising edge on a scalar signal. This option is ignored for compound signals.

- -falling

  (optional) Specifies a search for falling edge on a scalar signal. This option is ignored for compound signals.

- -anyedge

  (optional) Specifies a search for a rising or falling edge on a scalar signal. This option is ignored for compound signals. (default)

- -startDelta <num>

  (optional) Indicates a simulation delta cycle on which to start.

<num> — Any positive integer.

- -value <string>

  (optional) Specifies a match of a single scalar or compound signal against a specified string.

  <string> — Specifies a string to be matched.

- <startTime> [<unit>]

  (required) Specifies the simulation time at which to start the search. The time is specified as an integer or decimal number. Must be placed immediately before the **<pattern>** argument.

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

- <pattern>

  (Required unless the **-expr** argument is used.) Specifies one or more signal names or wildcard patterns of signal names to search on. Must be specified as the final argument to the **searchlog** command.

Related Topics

- virtual signal
- virtual log
- virtual nolog
- GUI_expression_format

see

This command displays the specified number of source file lines around the current execution line and places a marker to indicate the current execution line. If specified without arguments, five lines will be displayed before and four lines after.

Syntax

see [<n> | <pre> <post>]

Arguments

- <n>

  (optional) Designates the number of lines to display before and after the current execution line.

- <pre>

  (optional) Designates the number of lines to display before the current execution line.

- <post>

  (optional) Designates the number of lines to display after the current execution line.

Example

- Display 2 lines before and 5lines after the current execution line.

  see 2 5

  Returns:

```
#   92 :
#   93 :              if (verbose) $display("Read/Write test done");
# ->94 :              $stop(1);
#   95 :          end
#   96 :      end
#   97 :
#   98 :      or2 i1 (
#   99 :          .y(t_set),
```

setenv

This command changes or reports the current value of an environment variable. The setting is valid only for the current ModelSim session.

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

Syntax

setenv **<varname>** [<value>]

Arguments

- <varname>

  (required) The name of the environment variable you wish to set or check. Must be specified as the first argument to the **setenv** command.

- <value>

  (optional) The new value for <varname>. If you do not specify a value, ModelSim reports the variable's current value.

Related Topics

- unsetenv
- printenv

shift

This command shifts macro parameter values left one place, so that the value of parameter \$2 is assigned to parameter \$1, the value of parameter \$3 is assigned to \$2, and so on. The previous value of \$1 is discarded.

The **shift** command and macro parameters are used in macro files. If a macro file requires more than nine parameters, they can be accessed using the **shift** command.

To determine the current number of macro parameters, use the argc variable.

Syntax

shift

Arguments

- None

Description

For a macro file containing nine macro parameters defined as $1 to $9, one **shift** command shifts all parameter values one place to the left. If more than nine parameters are named, the value of the tenth parameter becomes the value of $9 and can be accessed from within the macro file.

Related Topics

- do

show

This command lists HDL objects and subregions visible from the current environment.

The objects listed include:

- **VHDL** — signals, processes, constants, variables, and instances.
- **Verilog —** nets, registers, tasks, functions, instances, variables, and memories.

The **show** command returns formatted results to stdout. To eliminate formatting (to use the output in a Tcl script), use the **Show** form of the command instead.

Syntax

show [-all] [<pathname>]

Arguments

- -all

  (optional) Displays all names at and below the specified path recursively.
- <pathname>

  (optional) Specifies the pathname of the environment for which you want the objects and subregions to be listed; if omitted, the current environment is assumed.

Examples

- List the names of all the objects and subregion environments visible in the current environment.

  show
- List the names of all the objects and subregions visible in the environment named /uut.

  show /uut
- List the names of all the objects and subregions visible in the environment named sub_region which is directly visible in the current environment.

  show sub_region
- List the names of all the objects and subregions visible in all top-level environments.

  show -all /

Related Topics

- environment
- find

simstats

This command returns performance-related statistics about elaboration and simulation. The statistics measure the simulation kernal process (vsimk) for a single invocation of vsim. If you invoke vsim a second time, or restart the simulation, the current statistics are discarded and new values are collected.

If executed without arguments, the command returns a list of pairs similar to the following:

```
{{elab memory} 0} {{elab working set} 7245824} {{elab time} 0.942645}
{{elab cpu time} 0.190274} {{elab context} 0} {{elab page faults} 1549}
{memory 0} {{working set} 0} {time 0} {{cpu time} 0} {context 0}
{{page faults} 0}
```

The elaboration statistics are measured one time at the end of elaboration. The simulation memory statistics are measured at the time you invoke **simstats**. The simulation time statistics are updated at the end of each run command. See the arguments below for descriptions of each statistic.

Units for time values are in seconds. Units for memory values vary by platform:

- For SunOS and Linux, the memory size is reported in Kbytes
- For Windows, the memory size is reported in bytes.

Some of the values may not be available on all platforms and other values may be approximates. Different operating systems report these numbers differently.

Syntax

simstats [memory | working | time | cpu | context | faults]

Arguments

- memory

  (optional) Returns the amount of virtual memory that the OS has allocated for vsimk.

- working

  (optional) Returns the portion of allocated virtual memory that is currently being used by vsimk. If this number exceeds the actual memory size, you will encounter performance degradation.

- time

  (optional) Returns the cumulative “wall clock” time of all run commands.

- cpu

  (optional) Returns the cumulative processor time of all run commands. Processor time differs from wall clock time in that processor time is only counted when the cpu is actually running vsimk. If vsimk is swapped out for another process, cpu time does not increase.

- context

  (optional) Returns the number of context swaps (vsimk being swapped out for another process) that occurred during all run commands.

- faults

  (optional) Returns the number of page faults that occurred during all run commands.

stack down

This command moves down the call stack.

If invoked without arguments, the command moves down the call stack by 1 level. The Locals window displays local variables at the level.

Syntax

stack down [n]

Arguments

- n

  (optional) Moves down the call stack by *n* levels. The default value is 1 level.

Related Topics

stack frame
stack level
stack tb
stack up

stack frame

This command selects the specified call frame.

Syntax

stack frame <n>

Arguments

- <n>

  Selects call frame number *n*. The currently executing frame is zero (also called the innermost) frame, frame one is the frame that called the innermost, and so on. The highest numbered frame is that of *main*.

Related Topics

stack down
stack level
stack tb
stack up

stack level

This command reports the current call frame number.

Syntax

stack level

Arguments

- None

Related Topics

stack down
stack frame
stack tb
stack up

stack tb

The **stack tb** command is an alias for the tb command.

Refer to the tb command for a complete syntax description.

stack up

This command moves up the call stack.

If invoked without arguments, the command moves up the call stack by 1 level. The Locals window displays local variables at the level.

Syntax

stack up [n]

Arguments

- n

  (optional) Moves up the call stack by *n* levels. The default value is 1 level.

Related Topics

stack down
stack frame
stack level
stack tb

status

This command lists summary information about currently interrupted macros.

If invoked without arguments, the command lists the filename of each interrupted macro, the line number at which it was interrupted, and prints the command itself. It also displays any onbreak or onerror commands that have been defined for each interrupted macro.

Syntax

status [file | line]

Arguments

- file

  (optional) Reports the file pathname of the current macro.

- line

  (optional) Reports the line number of the current macro.

Examples

The transcript below contains examples of resume, and **status** commands.

```
VSIM(paused)> status
# Macro resume_test.do at line 3 (Current macro)
#     command executing: "pause"
#     is Interrupted
#     ONBREAK commands: "resume"
# Macro startup.do at line 34
#     command executing: "run 1000"
#     processing BREAKPOINT
#     is Interrupted
#     ONBREAK commands: "resume"
VSIM(paused)> resume
# Resuming execution of macro resume_test.do at line 4
```

Related Topics

- abort
- do
- pause
- resume

step

The **step** command is an alias for the run command with the -step switch.

Refer to the run command for a complete syntax description.

stop

This command is used with the when command to stop simulation in batch files.

The **stop** command has the same effect as hitting a breakpoint. The **stop** command may be placed anywhere within the body of the when command.

Syntax

stop [-sync]

Arguments

- -sync

  (optional) Stops the currently running simulation at the next time step.

Description

Use run **-continue** to continue the simulation run, or the resume command to continue macro execution. If you want macro execution to resume automatically, put the **resume** command at the top of your macro file:

```
onbreak {resume}
```

Note

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

Related Topics

- bp
- resume
- run
- when

suppress

This command prevents one or more specified messages from displaying. You cannot suppress Fatal or Internal messages. The **suppress** command used without arguments returns the message numbers of all suppressed messages.

Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

Syntax

suppress [-clear <msg_number>[,<msg_number>,...]] [<msg_number>[,<msg_number>,…]] [<code_string>[, <code_string>,...]]

Arguments

- -clear <msg_number>[,<msg_number>,...]

  (optional) Clears suppression of one or more messages identified by message number.

  <msg_number> — A number identifying the message. Multiple message numbers are specified as a comma separated list.

- <msg_number>[,<msg_number>,...]

  (optional) A number identifying the message to be suppressed. Multiple message numbers are specified as a comma separated list.

- <code_string>[, <code_string>,...]

  (optional) A string identifier of the message to be suppressed. Disables warning messages in the category specified by <CODE>. Warnings that can be disabled include the <CODE> name in square brackets in the warning message.

Examples

- Return the message numbers of all suppressed messages:

  suppress

- Suppress messages by message number:

  suppress 8241,8242,8243,8446,8447

- Suppress messages by numbers and code categories:

  suppress 8241,TFMPC,CNNODP,8446,8447

- Clear message suppression for the designated messages:

  suppress -clear 8241,8242

- Return the message numbers of all suppressed messages and clear suppression for all:

  suppress -clear [suppress]

tb

This (traceback) command (traceback) displays a stack trace for the current process in the Transcript window. This lists the sequence of HDL function calls that have been entered to arrive at the current state for the active process.

Syntax

tb

Arguments

- None

Time

These commands allow you to perform comparisons between, operations on, and conversions of, time values.

When [unit] is left unspecified the commands assume the current simulation time unit, as defined by the Resolution variable in the *modelsim.ini* file or the vsim **-t** command. Units of time (ns, us, and so forth) can be specified independently for each <time[1 | 2]> specified for most of the commands. See the description of each command and examples for more information.

Arguments to this command are order dependent. Follow the order specified in the Syntax for each command.

Syntax

eqTime **<time1>[unit] <time2>[unit]**

Returns a 1 (true) or 0 (false) if <time1> and <time2> are equal.

neqTime **<time1>[unit] <time2>[unit]**

Returns a 1 (true) or 0 (false) if <time1> and <time2> are not equal.

ltTime **<time1>[unit] <time2>[unit]**

Returns a 1 (true) or 0 (false) if <time1> is less than <time2>.

gtTime **<time1>[unit] <time2>[unit]**

Returns a 1 (true) or 0 (false) if <time1> is greater than <time2>.

lteTime **<time1>[unit] <time2>[unit]**

Returns a 1 (true) or 0 (false) if <time1> is less than or equal to <time2>.

gteTime **<time1>[unit] <time2>[unit]**

Returns a 1 (true) or 0 (false) if <time1> is greater than or equal to <time2>.

addTime **<time1>[unit] <time2>[unit]**

Returns the sum of adding <time1> to <time2>

subTime **<time1>[unit] <time2>[unit]**

Returns the value of subtracting <time2> from <time1>

mulTime **<time1>[unit] <integer>**

Returns the value of multiplying <time1> by an <integer>

divTime **<time1>[unit] <time2>[unit]**

Returns an integer, that is the value of dividing <time1> by <time2>. Specifying 0 for <time2> results in an error.

intToTime **<high_32bit_int> <low_32bit_int>**

Returns a 64-bit time value based on two 32-bit parts of a 64-bit integer. This command is useful when you've performed an integer calculation that results in a 64-bit value and need to convert it to a time unit.

scaleTime **<time1>[unit] <scale_factor>**

Returns a time value scaled by a real number and truncated to the current time resolution.

RealToTime **<real>**

Returns a time value equivalent to the specified real number and truncated to the current time resolution.

validTime **<time>**

Returns a 1 (true) or 0 (false) if the given string is a valid time for use with any of these Time calculations.

formatTime **{+ | -} commas | {+ | -}nodefunit | {+ | -}bestunits**

Sets display properties for time values.

Arguments

- <time1>[unit] ...

  <time> — Specified as an integer or decimal number. Current simulation units are the default unless specifying **<unit>**.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting <unit>. Valid VHDL time units are:

  fs — femtosecond (10^{-15} seconds)

  ps — picosecond (10^{-12} seconds)

  ns — nanosecond (10^{-9} seconds)

  us — microsecond (10^{-6} seconds)

  ms — millisecond (10^{-3} seconds)

  sec — second

  min — minute (60 seconds)

  hr — hour (3600 seconds)

  Note that if you put a space between the values, you must enclose the argument in braces ({ }) or quotation marks (" ").

- <high_32bit_int> | <low_32bit_int>

  <high_32bit_int> — The "high" part of the 64-bit integer.

  <low_32bit_int> — The "low" part of the 64-bit integer.

- <scale_factor> — a real number to be used as scaling factor. Common values can include:

  ```
  0.25, 0.5, 1.5, 2, 10, 100
  ```

- {+ | -} commas — controls whether commas are displayed in time values.

  +commas — time values include commas

  -commas — time values do not include commas

- {+ | -}nodefunit — controls whether time values display time units

  +nodefunit — time values do not include time units and will be in current time resolution

  -nodefunit — time values may include time units

- {+ | -}bestunits — controls whether time values display the largest possible time unit, for example 8 us instead of 8,000 ns.

  +bestunits — time values display the largest possible time unit

  -bestunits — time values display the default time unit

Examples

>ltTime 100ns 1ms

Returns:

```
# 1
```

>addTime {1545 ns} {455 ns}

Returns:

```
# 2 us
```

>gteTime "1000 ns" "1 us"

Returns:

```
# 1
```

>divTime 1us 10ns

Returns:

```
# 100
```

>formatTime +bestunit

Returns:

```
# -commas -nodefunit +bestunits
```

>scaleTime 3ms 1000

Returns:

```
# 3 sec
```

>RealToTime 1.345e04

Returns:

```
# 13450 ns
```

transcript

This command controls echoing of commands executed in a macro file.

If no option is specified, the current setting is reported.

Syntax

transcript [on | off | -q | quietly]

Arguments

- on

  (optional) Specifies that commands in a macro file will be echoed to the Transcript window as they are executed.

- off

  (optional) Specifies that commands in a macro file will not be echoed to the Transcript window as they are executed.

- -q

  (optional) Returns "0" if transcripting is turned off or "1" if transcripting is turned on. Useful in a Tcl conditional expression.

- quietly

  (optional) Turns off the transcript echo for all commands. To turn off echoing for individual commands see the quietly command.

Examples

- Commands within a macro file will be echoed to the Transcript window as they are executed.

  transcript on

- If issued immediately after the previous example, the message:

  transcript

  Returns

  ```
  Macro transcripting is turned ON.
  ```

Related Topics

- Transcript Window
- echo

transcript file

This command sets or queries the current PrefMain(file) Tcl preference variable. You can use this command to clear a transcript in batch mode or to limit the size of a transcript file. It offers an alternative to setting the PrefMain(file) Tcl preference variable through the GUI.

Syntax

transcript file [<filename>]

Arguments

- <filename>

  (optional) Specifies a name for the transcript file. Wildcard characters are allowed, and "stdout" or "stderr" are valid file names. If you specify a new file, the existing transcript file is closed and a new transcript file opened. If you specify an empty string (""), the existing file is closed and no new file is opened. If you don't specify this argument, the current filename is returned.

Note

You can prevent overwriting older transcript files by including a pound sign (#) in **<filename>** when **<filename>** is a repeated string. The simulator replaces the pound character (#) with the next available sequence number when saving a new transcript file.

Examples

- Close the current transcript file and stops writing data to the file. This is a method for reducing the size of your transcript.

  ```
  transcript file ""
  ```

- Close the current transcript file named *trans1.txt and* open a new transcript file, incrementing the file name by 1.

  ```
  transcript file trans#.txt
  ```

  Closes *trans1.txt* and opens *trans2.txt.*

- This series of commands results in the transcript containing only data from the second millisecond of the simulation. The first **transcript file** command closes the transcript so no data is being written to it. The second **transcript file** command opens a new transcript and records data from 1 ms to 2 ms.

  ```
  transcript file ""
  run 1 ms
  transcript file transcript
  run 1 ms
  ```

Related Topics

- "Creating a Transcript File"
- "Setting Preference Variables from the GUI"
- Transcript Window
- transcript path
- transcript sizelimit

transcript path

This command returns the full pathname to the current transcript file.

Syntax

transcript path

Arguments

- None

Related Topics

- "Creating a Transcript File"
- "Setting Preference Variables from the GUI"
- “Transcript Window”
- transcript file

transcript sizelimit

This command sets or queries the current preference value for the transcript fileSizeLimit value. If the size limit is reached, the transcript file is saved and simulation continues.

Syntax

transcript sizelimit [<size>]

Arguments

- <size>

  (optional) Specifies the size, in KB, of the transcript file where the default is 0 or unlimited. The actual file size may be larger by as much as one buffer size (usually about 4k), depending on the operating system default buffer size and the size of the messages written to the transcript.

Note

You can set the size of the transcript file with the $PrefMain (fileSizeLimit) Tcl variable in the Preferences dialog. Refer to "Setting Preference Variables from the GUI" for more information.

Related Topics

- "Creating a Transcript File"
- "Setting Preference Variables from the GUI"
- “Transcript Window”
- transcript file

tssi2mti

This command is used to convert a vector file in TSSI Format into a sequence of force and run commands.

The stimulus is written to the standard output.

The source code for **tssi2mti** is provided in the *examples* directory as:

```
<install_dir>/examples/tssi2mti/tssi2mti.c
```

Syntax

tssi2mti **<signal_definition_file>** [<sef_vector_file>]

Arguments

- <signal_definition_file>

  (required) Specifies the name of the TSSI signal definition file describing the format and content of the vectors.

- <sef_vector_file>

  (optional) Specifies the name of the file containing vectors to be converted. If none is specified, standard input is used.

Examples

- The command will produce a do file named *trigger.do* from the signal definition file *trigger.def* and the vector file *trigger.sef*.

  tssi2mti trigger.def trigger.sef > trigger.do

- This example is the same as the previous one, but uses the standard input instead.

  tssi2mti trigger.def < trigger.sef > trigger.do

Related Topics

- force
- run
- write tssi

ui_VVMode

This command specifies behavior when encountering UI registration calls used by verification packages, such as AVM or OVM. Returns the current setting when specifies without an argument.

Syntax

ui_VVMode [full | logclass | logobj | nolog | off]

Arguments

- full

  (optional) Enables the context registration of the UI registration call and automatically logs both the class type and the registered object to the WLF file.

- logclass

  (optional) Enables the context registration of the UI registration call and automatically logs the class type of the registered object to the WLF file.

- logobj

  (optional) Enables the context registration of the UI registration call and automatically logs the registered object to the WLF file

- nolog

  (optional) Enables the context registration of the UI registration call, but does not automatically log the registration to the WLF file. (default)

- off

  (optional) Disables context registration and automatic logging when encountering UI registration calls.

Description

UI registration calls, Verilog system tasks specific to this product, are typically included in verification packages such as AVM and OVM so that key information about the packages is available when debugging the simulation. The UI registration calls include:

- $ui_VVInstallInst() — Defines a region in the context tree, which will appear in the Structure window.
- $ui_VVInstallObj() — Adds an object to the defined parent, which will appear in the Objects window when the parent instance is selected in the Structure window.
- $ui_VVInstallPort() — Adds a port that is an object that connects to another component, which will appear in the Objects window when the parent instance is selected in the Structure window.
- $ui_VVSetFilter() — Specifies which class properties should not be shown in the GUI.

- $ui_VVSetAllow() — Specifies which class properties should be retained that were filtered out with $ui_VVSetFilter.

unsetenv

This command deletes an environment variable. The deletion is not permanent – it is valid only for the current ModelSim session.

Syntax

unsetenv **<varname>**

Arguments

- <varname>

  (required) The name of the environment variable you wish to delete.

Related Topics

- setenv
- printenv

vcd add

This command adds the specified objects to a VCD file.

The allowed objects are Verilog nets and variables and VHDL signals of type bit, bit_vector, std_logic, and std_logic_vector (other types are silently ignored). The command works with mixed HDL designs.

All **vcd add** commands must be executed at the same simulation time. The specified objects are added to the VCD header and their subsequent value changes are recorded in the specified VCD file. By default all port driver changes and internal variable changes are captured in the file. You can filter the output using arguments detailed below.

Related Verilog tasks: $dumpvars, $fdumpvars

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

Syntax

vcd add [-dumpports] [-file <filename>] [[-in] [-out] [-inout] | [-ports]] [-internal] [-r | -r -optcells] **<object_name> ...**

Arguments

- -dumpports

  (optional) Specifies port driver changes to be added to an extended VCD file. When the **vcd dumpports** command cannot specify all port driver changes that will appear within the VCD file, multiple **vcd add -dumpports** commands can be used to specify additional port driver changes.

- -file <filename>

  (optional) Specifies the name of the VCD file. This option should be used only when you have created multiple VCD files using the vcd files command.

  <filename> — A *.vcd* file.

- -in

  (optional) Includes only port driver changes from ports of mode IN.

- -out

  (optional) Includes only port driver changes from ports of mode OUT.

- -inout

  (optional) Includes only port driver changes from ports of mode INOUT.

- -ports

  (optional) Includes only port driver changes. Excludes internal variable or signal changes.

- -internal

  (optional) Includes only internal variable or signal changes. Excludes port driver changes.

- -r | -r -optcells

  (optional) Specifies that signal and port selection occurs recursively into subregions. If omitted, included signals and ports are limited to the current region. When -r is used with -optcells it allows Verilog optimized cell ports to be visible when using wildcards. By default Verilog optimized cell ports are not selected even if they match the specified wildcard pattern.

- <object_name> ...

  (required) Specifies the Verilog or VHDL object or objects to add to the VCD file. Multiple objects may be specified by separating names with spaces. Wildcards are accepted. Must be specified as the final argument to the **vcd add** command.

Related Topics

- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd checkpoint

This command dumps the current values of all VCD variables to the specified VCD file. While simulating, only value changes are dumped.

Related Verilog tasks: $dumpall, $fdumpall

Syntax

vcd checkpoint [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the VCD file. If omitted the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- vcd add
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd comment

This command inserts the specified comment in the specified VCD file.

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

Syntax

vcd comment **<comment string>** [<filename>]

Arguments

- <comment string>

  (required) Comment to be included in the VCD file. Must be enclosed by double quotation marks or curly braces. Must be specified as the first argument to the **vcd comment** command.

- <filename>

  (optional) Specifies the name of the VCD file. If omitted the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- vcd add
- vcd checkpoint
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd dumpports

This command creates a VCD file that includes port driver data.

By default all port driver changes are captured in the file. You can filter the output using arguments detailed below. Related Verilog task: $dumpports

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

Syntax

vcd dumpports [-compress] [-direction] [-file <filename>] [-force_direction] [-in] [-out] [-inout]
[-no_strength_range] [-unique] [-vcdstim] **<object_name> ...**

Arguments

- -compress

  (optional) Produces a compressed VCD file. ModelSim uses the gzip compression algorithm. It is not necessary to specify **-compress** if you specify a *.gz* extension with the **-file <filename>** argument

- -direction

  (optional) Includes driver direction data in the VCD file.

- -file <filename>

  (optional) Creates a VCD file. Defaults to the current working directory and the filename *dumpports.vcd*. Multiple filenames can be opened during a single simulation.

  <filename> — Specifies a filename. When specified with a *.gz* extension, the file is compressed.

- -force_direction

  (optional) Causes **vcd dumpports** to use the specified port direction (instead of driver location) to determine whether the value being dumped is input or output. This argument overrides the default use of the location of drivers on the net to determine port direction (this is because Verilog port direction is not enforced by the language or by ModelSim).

- -in

  (optional) Includes ports of mode IN.

- -out

  (optional) Includes ports of mode OUT.

- -inout

  (optional) Includes ports of mode INOUT.

- -no_strength_range

  (optional) Ignores strength ranges when resolving driver values. This argument is an extension to the IEEE 1364 specification. Refer to Resolving Values for additional information.

- -unique

  (optional) Generates unique VCD variable names for ports even if those ports are connected to the same collapsed net.

- -vcdstim

  (optional) Ensures that port name order in the VCD file matches the declaration order in the instance module or entity declaration. Refer to Port Order Issues for further information.

- <object_name> ...

  (required) Specifies one or more Verilog, VHDL, or SystemC objects to add to the VCD file. You can specify multiple objects by separating names with spaces. Wildcards are accepted. Must be specified as the final argument to the **vcd dumpports** command.

Examples

- Create a VCD file named *counter.vcd* of all IN ports in the region /test_design/dut/.

  ```
  vcd dumpports -in -file counter.vcd /test_design/dut/*
  ```

- These two commands resimulate a design from a VCD file. Refer to Simulating with Input Values from a VCD File for further details.

  ```
  vcd dumpports -file addern.vcd /testbench/uut/*
  vsim -vcdstim addern.vcd addern -gn=8 -do "add wave /*; run 1000"
  ```

- This series of commands creates VCD files for the instances *proc* and *cache* and then re-simulates the design using the VCD files in place of the instance source files. Refer to Replacing Instances with Output Values from a VCD File for more information.

  ```
  vcd dumpports -vcdstim -file proc.vcd /top/p/*
  vcd dumpports -vcdstim -file cache.vcd /top/c/*
  run 1000

  vsim top -vcdstim /top/p=proc.vcd -vcdstim /top/c=cache.vcd
  ```

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd dumpportsall

This command creates a checkpoint in the VCD file which shows the value of all selected ports at that time in the simulation, regardless of whether the port values have changed since the last timestep.

Related Verilog task: $dumpportsall

Syntax

vcd dumpportsall [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd dumpportsflush

This command flushes the contents of the VCD file buffer to the specified VCD file.

Related Verilog task: $dumpportsflush

Syntax

vcd dumpportsflush [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd dumpportslimit

This command specifies the maximum size of the VCD file (by default, limited to available disk space). When the size of the file exceeds the limit, a comment is appended to the file and VCD dumping is disabled.

Related Verilog task: $dumpportslimit

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

Syntax

vcd dumpportslimit **<dumplimit>** [<filename>]

Arguments

- <dumplimit>

  (required) Specifies the maximum VCD file size in bytes. Must be specified as the first argument to the **vcd dumpportslimit** command.

- <filename>

  (optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd dumpportsoff

This command turns off VCD dumping and records all dumped port values as x.

Related Verilog task: $dumpportsoff

Syntax

vcd dumpportsoff [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd dumpportson

This command turns on VCD dumping and records the current values of all selected ports. This command is typically used to resume dumping after invoking vcd dumpportsoff.

Related Verilog task: $dumpportson

Syntax

vcd dumpportson [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the VCD file. If omitted the command is executed on all open VCD files.

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd file

This command specifies the filename and state mapping for the VCD file created by a vcd add command. The **vcd file** command is optional. If used, it must be issued before any **vcd add** commands.

Related Verilog task: $dumpfile

Syntax

vcd file [-dumpports] [-direction] [<filename>] [-map <mapping pairs>] [-no_strength_range] [-nomap] [-unique]

Arguments

- -dumpports

  (optional) Capture detailed port driver data for Verilog ports and VHDL std_logic ports. This option works only on ports, and any subsequent **vcd add** command will accept only qualifying ports (silently ignoring all other specified objects).

- -direction

  (optional) Includes driver direction data in the VCD file.

- <filename>

  (optional) Specifies the name of the VCD file that is created where the default is *dump.vcd*.

- -map <mapping pairs>

  (optional) Overrides the default mappings. Affects only VHDL signals of type std_logic.

  <mapping pairs> — Specified as a list of character pairs. The first character in a pair must be one of the std_logic characters UX01ZWLH- and the second character is the character you wish to be recorded in the VCD file. The Tcl convention for command strings that include spaces is to enclose them in quotation marks (" "). For example, to map L and H to z:

  vcd file -map "L z H z"

- -no_strength_range

  (optional) Ignores strength ranges when resolving driver values. This argument is an extension to the IEEE 1364 specification. Refer to Resolving Values for additional information.

- -nomap

  (optional) Affects only VHDL signals of type std_logic. It specifies that the values recorded in the VCD file shall use the std_logic enumeration characters of UX01ZWLH-. This option results in a non-standard VCD file because VCD values are limited to the four state character set of x01z. By default, the std_logic characters are mapped as follows.

VHDL	VCD	VHDL	VCD
U	x	W	x
X	x	L	0
0	0	H	1
1	1	-	x
Z	z		

- -unique

  (optional) Generates unique VCD variable names for ports even if those ports are connected to the same collapsed net.

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd files

This command specifies filenames and state mapping for VCD files created by the **vcd add** command. The **vcd files** command is optional. If used, it must be issued before any **vcd add** commands.

Related Verilog task: $fdumpfile

Syntax

vcd files [-compress] [-direction] <filename> [-map <mapping pairs>] [-no_strength_range] [-nomap] [-unique]

Arguments

- -compress

  (optional) Produces a compressed VCD file. ModelSim uses the gzip compression algorithm. If you specify a *.gz* extension on the **-file <filename>** argument, ModelSim compresses the file even if you don't use the **-compress** argument.

- -direction

  (optional) Includes driver direction data in the VCD file.

- <filename>

  (required) Specifies the name of a VCD file to create. Multiple files can be opened during a single simulation; however, you can create only one file at a time. If you want to create multiple files, invoke **vcd files** multiple times.

- -map <mapping pairs>

  (optional) Overrides the default mappings. Affects only VHDL signals of type std_logic.

  <mapping pairs> — Specified as a list of character pairs. The first character in a pair must be one of the std_logic characters UX01ZWLH- and the second character is the character you wish to be recorded in the VCD file. The Tcl convention for command strings that include spaces is to enclose them in quotation marks (" "). For example, to map L and H to z:

  ```
  vcd file -map "L z H z"
  ```

- -no_strength_range

  (optional) Ignores strength ranges when resolving driver values. This argument is an extension to the IEEE 1364 specification. Refer to "Resolving Values" for additional information.

- -nomap

  (optional) Affects only VHDL signals of type std_logic. It specifies that the values recorded in the VCD file shall use the std_logic enumeration characters of UX01ZWLH-. This option

results in a non-standard VCD file because VCD values are limited to the four state character set of x01z. By default, the std_logic characters are mapped as follows.

VHDL	VCD	VHDL	VCD
U	x	W	x
X	x	L	0
0	0	H	1
1	1	-	x
Z	z		

- -unique

  (optional) Generates unique VCD variable names for ports even if those ports are connected to the same collapsed net.

Examples

The following example shows how to "mask" outputs from a VCD file until a certain time after the start of the simulation. The example uses two **vcd files** commands and the vcd on and vcd off commands to accomplish this task.

```
vcd files in_inout.vcd
vcd files output.vcd
vcd add -in -inout -file in_inout.vcd /*
vcd add -out -file output.vcd /*
vcd off output.vcd
run 1us
vcd on output.vcd
run -all
```

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd flush
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd flush

This command flushes the contents of the VCD file buffer to the specified VCD file. This command is useful if you want to create a complete VCD file without ending your current simulation.

Related Verilog tasks: $dumpflush, $fdumpflush

Syntax

vcd flush [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the VCD file. If omitted, the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd limit
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd limit

This command specifies the maximum size of a VCD file (by default, limited to available disk space).

When the size of the file exceeds the limit, a comment is appended to the file and VCD dumping is disabled.

Related Verilog tasks: $dumplimit, $fdumplimit

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

Syntax

vcd limit **<filesize>** [<filename>]

Arguments

- <filesize>

  (Required) Specifies the maximum VCD file size, in bytes. The numerical value of <filesize> can only be a whole number. Must be specified as the first argument to the **vcd limit** command.

  You can specify a unit of Kb, Mb, or Gb with the numerical value (units are case insensitive). Do not insert a space between the numerical value and the unit (for example, 400Mb, not 400 Mb).

- <filename>

  (Optional) Specifies the name of the VCD file. If omitted, the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

Example

- Specify a maximum VCD file size of 6 gigabytes and a VCD file named my_vcd_file.vcd.

  ```
  vcd limit 6gb my_vcd_file.vcd
  ```

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd off
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd off

This command turns off VCD dumping to the specified file and records all VCD variable values as x.

Related Verilog tasks: $dumpoff, $fdumpoff

Syntax

vcd off [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the VCD file. If omitted, the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd on
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd on

This command turns on VCD dumping to the specified file and records the current values of all VCD variables.

By default, **vcd on** is automatically performed at the end of the simulation time that the vcd add command performed.

Related Verilog tasks: $dumpon, $fdumpon

Syntax

vcd on [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the VCD file. If omitted the command is executed on the file designated by the vcd file command or *dump.vcd* if **vcd file** was not invoked.

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd2wlf
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcd2wlf

This command is a utility that translates a VCD (Value Change Dump) file into a WLF file that you can display in ModelSim using the **vsim -view** argument. This command only works on VCD files containing positive time values.

Description

The **vcd2wlf** command functions as simple one-pass converter. If you are defining a bus in a VCD file, you must specify all bus bits before the next $scope or $upscope statement appears in the file. The best way to ensure that bits get converted together as a bus is to declare them on consecutive lines.

For example:

```
Line 21 : $var wire 1 $ in [2] $end
Line 22 : $var wire 1 $u in [1] $end
Line 23 : $var wire 1 # in [0] $end
```

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

Syntax

vcd2wlf [-splitio] [-splitio_in_ext <extension>] [-splitio_out_ext <extension>] [-nocase] {<vcd filename> | – } <wlf filename>

Arguments

- -splitio

  (optional) Specifies that extended VCD port values are to be split into their corresponding input and output components by creating two signals instead of just one in the resulting *.wlf* file. By default the new input-component signal keeps the same name as the original port name while the output-component name is the original name with "__o" appended to it.

- -splitio_in_ext <extension>

  (optional) Adds an extension to input-component signal names created by using **-splitio**.

  <extension> — Specifies a string.

- -splitio_out_ext <extension>

  (optional) Adds an extension to output-component signal names created by using **-splitio**.

  <extension> — Specifies a string.

- -nocase

  (optional) Converts all alphabetic identifiers to lowercase.

- {<vcd filename> | – }

  (required) Specifies the name of the VCD file, or standard input (-), you want to translate into a WLF file. Must be specified immediately preceding the <wlf filename> argument to the **vcd2wlf** command.

- <wlf filename>

  (required) Specifies the name of the output WLF file. Must be specified as the final argument to the **vcd2wlf** command.

Example

- Concatenate *my.vcd* file and pipe standard input to **vcd2wlf** and save output to *my.wlf* file.

  ```
  cat my.vcd | vcd2wlf - my.wlf
  ```

- Redirect input from the file *my.vcd* file to **vcd2wlf** and save the output to *my.wlf* file.

  ```
  vcd2wlf - my.wlf  <my.vcd
  ```

Related Topics

- vcd add
- vcd checkpoint
- vcd comment
- vcd dumpports
- vcd dumpportsall
- vcd dumpportsflush
- vcd dumpportslimit
- vcd dumpportsoff
- vcd dumpportson
- vcd file
- vcd files
- vcd flush
- vcd limit
- vcd off
- vcd on
- DumpportsCollapse
- Value Change Dump (VCD) Files
- Verilog tasks are documented in the Language Reference Manual IEEE 1364 standard.

vcom

The **vcom** command compiles VHDL source code into a specified working library (or to the **work** library by default).

You can invoke **vcom** either from within ModelSim or from the command prompt of your operating system. You can invoke this command during simulation.

Compiled libraries are dependent on the major version of ModelSim. When moving between major versions, you must refresh compiled libraries using the **-refresh** argument to **vcom**. This is not required for minor versions (letter releases).

All arguments to the **vcom** command are case-sensitive. For example, -WORK and -work are not equivalent.

This command provides additional information with the -help switch.

Syntax

vcom [options] <filename> [<filename> ...]

[options]:

[-87 | -93 | -2002 | -2008]

[-addpragmaprefix <prefix>]
[-allowProtectedBeforeBody] [-amsstd | -noamsstd]

[-bindAtCompile] [-bindAtLoad]

[-check_synthesis]

[-debugVA] [-deferSubpgmCheck | -noDeferSubpgmCheck]

[-error <msg_number>[,<msg_number>,…]] [-explicit]

[-f <filename>] [-fatal <msg_number>[,<msg_number>,…]]
[-force_refresh <design_unit>] [-fsmimplicittrans | -nofsmimplicittrans]
[-fsmresettrans | -nofsmresettrans] [-fsmsingle | -nofsmsingle]
[-fsmverbose [b | t | w]]

[-gen_xml <design_unit> <filename>]

[-ignoredefaultbinding] [-ignorepragmaprefix <prefix>] [ignoreStandardRealVector]
[-ignorevitalerrors] [-initoutcompositeparam | -noinitoutcompositeparam]

[-just abcep]

[-l <filename>] [-line <number>] [-lint] [-lower] [-lrmVHDLConfigVis]

[-mixedsvvh [b | l | r][i]] [-modelsimini <ini_filepath>]

[-no1164] [-noaccel <package_name>] [-nocasestaticerror] [-nocheck] [-nodbgsym]
[]-nofprangecheck [-noFunctionInline] [-noindexcheck] [-nologo] [-nonstddriverinit]
[-noothersstaticerror]
[-note <msg_number> [,<msg_number>, …]] [-novital] [-novitalcheck]
[-nowarn <category_number>]

[-oldconfigvis]

[-pedanticerrors] [-performdefaultbinding] [-preserve]

[-quiet]

[-rangecheck | -norangecheck] [-refresh]

[-s] [-separateConfigLibrary] [-skip abcep] [-source]
[-suppress <msg_number>[,<msg_number>,…]]

[-time]

[-version] [-vmake]

[-warning <msg_number>[,<msg_number>,…]] [-work <library_name>]

Arguments

- -87 | -93 | -2002 | -2008

  (optional) Specifies which LRM-specific compiler to use. You can also control this behavior with the VHDL93 variable in the *modelsim.ini* file. Refer to "Differences Between Versions of VHDL" for more information.

  -87 — Enables support for VHDL 1076-1987.

  -93 — Enables support for VHDL 1076-1993.

  -2002 — Enables support for VHDL 1076-2002. (default)

  -2008 — Enables support for VHDL 1076-2008.

- -addpragmaprefix <prefix>

  (optional) Enables recognition of synthesis and coverage pragmas with a user specified prefix. If this argument is not specified, pragmas are treated as comments. All regular synthesis and coverage pragmas are honored.

  <prefix> — Specifies a user defined string where the default is no sting, indicated by quotation marks.

  You may also set this with the AddPragmaPrefix variable in the vcom section of the *modelsim.ini* file.

- -allowProtectedBeforeBody

  (optional) Allows a variable of a protected type to be created prior to declaring the body.

- -amsstd | -noamsstd

  (optional) Specifies whether vcom adds the declaration of REAL_VECTOR to the STANDARD package. This is useful for designers using VHDL-AMS to test digital parts of their model.

  -amsstd — REAL_VECTOR is included in STANDARD.

  -noamsstd — REAL_VECTOR is not included in STANDARD (default).

  You can also control this with the AmsStandard variable or the MGC_AMS_HOME environment variable.

- -bindAtCompile

  (optional) Forces ModelSim to perform default binding at compile time rather than at load time. Refer to “Default Binding” for more information. You can change the permanent default by editing the BindAtCompile variable in the *modelsim.ini*.

- -bindAtLoad

  (optional) Forces ModelSim to perform default binding at load time rather than at compile time. (Default)

- -check_synthesis

  (optional) Turns on limited synthesis rule compliance checking. Specifically, it checks to see that signals read by a process are in the sensitivity list. The checks understand only combinational logic, not clocked logic. Edit the CheckSynthesis variable in the *modelsim.ini* file to set a permanent default.

- -debugVA

  (optional) Prints a confirmation if a VITAL cell was optimized, or an explanation of why it was not, during VITAL level-1 acceleration.

- -deferSubpgmCheck

  (optional) Forces the compiler to report array indexing and length errors as warnings (instead of as errors) when encountered within subprograms. Subprograms with indexing and length errors that are invoked during simulation cause the simulator to report errors, which can potentially slow down simulation because of additional checking.

- -error <msg_number>[,<msg_number>,…]

  Optional) Changes the severity level of the specified message(s) to "error." Edit the error variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

  <msg_number> — A number identifying the message. Multiple message numbers are specified as a comma separated list.

- -explicit

  (optional) Directs the compiler to resolve ambiguous function overloading by favoring the explicit function definition over the implicit function definition. 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>

  (optional) Specifies a file with more command-line arguments. Allows complex argument strings to be reused without retyping. Allows gzipped input files. Nesting of **-f** options is allowed.

  <filename> —

  Refer to the section "Argument Files" for more information.

- -fatal <msg_number>[,<msg_number>,…]

  (optional) Changes the severity level of the specified message(s) to "fatal." Edit the fatal variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

  <msg_number> — A number identifying the message. Multiple message numbers are specified as a comma separated list.

- -force_refresh <design_unit>

  (optional) Forces the refresh of all specified design units. By default, the work library is updated; use **-work <library_name>**, in conjunction with **-force_refresh**, to update a different library (for example, **vcom -work <your_lib_name> -force_refresh**).

  <design_unit> —

  When the compiler refreshes a design unit, it checks each dependency to ensure its source has not been changed and recompiled. Sometimes the dependency checking algorithm changes from release to release. This can lead to false errors during the integrity checks performed by the **-refresh** argument. An example of such a message follows:

  ```
  ** Error: (vsim-13) Recompile /u/test/dware/dware_61e_beta.dwpackages
  because /home/users/questasim/linux/../synopsys.attributes has changed.
  ```

  The **-force_refresh** argument forces the refresh of the design unit, overriding any dependency checking errors encountered by the **-refresh** argument.

  A more conservative approach to working around **-refresh** dependency checks is to recompile the source code, if it is available.

- -fsmimplicittrans | -nofsmimplicittrans

  (optional) Toggles recognition of implied same state transitions, which is off by default (-nofsmimplicittrans).

- -fsmresettrans | -nofsmresettrans

  (optional) Toggles recognition of sychronous or asynchronous reset transitions, and is on by default (-fsmresettrans). This includes/excludes reset transitions in coverage results.

- -fsmsingle | -nofsmsingle

  (optional) Toggles the recognition of VHDL FSMs where the current state variable of type std_logic, bit, boolean, or single-bit std_logic_vector/bit_vector and Verilog single-bit FSMs.

- -fsmverbose [b | t | w]

  (optional) Provides information about FSMs detected, including state reachability analysis.

  b — displays only basic information.

  t — displays a transition table in addition to the basic information.

  w — displays any warning messages in addition to the basic information.

  When you do not specify an argument, this switch reports all information similar to:

```
# ** Note: (vcom-1947)   FSM RECOGNITION INFO
#      Fsm detected in : ../fpu/rtl/vhdl/serial_mul.vhd
#      Current State Variable : s_state :
../fpu/rtl/vhdl/serial_mul.vhd(76)
#      Clock : clk_i
#      Reset States are: { waiting , busy }
#      State Set is : { busy , waiting }
#      Transition table is
#         ----------------------------------------
#         busy      =>   waiting Line : (114 => 114)
#         busy      =>   busy    Line : (111 => 111)
#         waiting   =>   waiting Line : (120 => 120) (114 => 114)
#         waiting   =>   busy    Line : (111 => 111)
#         ----------------------------------------
```

When you do not specify this switch, you will receive a message similar to:

```
# ** Note: (vcom-143) Detected '1' FSM/s in design unit
'serial_mul.rtl'.
```

- -gen_xml <design_unit> <filename>

  (optional) Produces an XML-tagged file containing the interface definition of the specified entity.

  <design_unit> — The name of an entity or design unit in the Work library. Wildcards and multiple design unit names are not allowed.

  <filename> — A user-specified name for the file.

  For example:

  This option requires a two-step process where you must:

  1) compile <filename> into a library with **vcom** (without **-gen_xml**) then

  2) execute **vcom** with the **-gen_xml** switch.

  ```
  vlib work
  vcom counter.vhd
  vcom -gen_xml counter counter.xml
  ```

- -ignoredefaultbinding

  (optional) Instructs the compiler not to generate a default binding during compilation. You must explicitly bind all components in the design to use this switch.

- -ignorepragmaprefix <prefix>

  (optional) Directs vcom to ignore synthesis and coverage pragmas with the specified prefixname. All affected pragmas will be treated as regular comments. Edit the IgnorePragmaPrefix *modelsim.ini* variable to set a permanent default.

  <prefix> — Specifies a user defined string.

- ignoreStandardRealVector

  (optional) Instructs ModelSim to ignore the REAL_VECTOR declaration in package STANDARD when compiling with **vcom -2008**. Edit the ignoreStandardRealVector *modelsim.ini* variable to set a permanent default. For more information refer to the REAL_VECTOR section in **Help > Technotes > vhdl2008migration**.

- -ignorevitalerrors

  (optional) Directs the compiler to ignore VITAL compliance errors. The compiler still reports that VITAL errors exist, but it will not stop the compilation. You should exercise caution in using this switch; as part of accelerating VITAL packages, we assume that compliance checking has passed.

- -initoutcompositeparam

  (optional) Causes initialization of subprogram parameters for array and record types when the subprogram is executed in designs compiled with LRM 1076-2002 and earlier. This argument forces the output parameters to their default initial ("left") values when entering a subprogram. By default, **-initoutcompositeparam** is enabled for designs compiled with **vcom -2008** and later. You can also enable this by setting the InitiOutCompositeParam variable to 1 in the modelsim.ini file.

- -noinitoutcompositeparam

  (optional) Disables initialization of subprogram parameters for array and record types when the subprogram is executed in designs compiled with LRM 1076-2002 and earlier. By default, designes compiled with LRM 1076-2008 and later do not initialize subprogram parameters for array and record types when the subprogram is executed. You can also disable initialization of subprogram parameters for array and record types by setting the InitiOutCompositeParam variable to 2 in the modelsim.ini file.

- -just abcep

  (optional) Directs the compiler to include only the following:

  a — architectures

  b — bodies

  c — configurations

  e — entities

  p — packages

  Any combination in any order can be used, but you must specify at least one choice if you use this switch.

- -l <filename>

  (optional) Generates a log file of the compile.

- -line <number>

  (optional) Starts the compiler on the specified line in the VHDL source file. By default, the compiler starts at the beginning of the file.

  <number> —

- -lint

  (optional) Performs additional static checks on case statement rules and enables warning messages for the following situations:

- The result of the built-in concatenation operator ("&") is the actual for a subprogram formal parameter of an unconstrained array type.
- If you specify the **-BindAtCompile** switch with **vcom,** the entity to which a component instantiation is bound has a port that is not on the component, and for which there is no error otherwise.
- A direct recursive subprogram call.
- In cases involving class SIGNAL formal parameters, as described in the IEEE Standard VHDL Language Reference Manual entitled "Signal parameters". This check only applies to designs compiled using **-87**. If you compile using **-93**, it would be flagged as a warning or error, even without the **-lint** argument. Can also be enabled using the Show_Lint variable in the modelsim.ini file.

- -lower

  (optional) Forces **vcom** to convert uppercase letters in object identifiers to lowercase. You can also enable this by setting the PreserveCase variable to 0 in the modelsim.ini file.

- -lrmVHDLConfigVis

  (optional, default) Forces vcom to use visibility rules that comply with the Language Reference Manual when processing VHDL configurations. Refer to **vcom -oldconfigvis** or the oldVHDLConfigurationVisibility variable in the modelsim.ini file for more information.

- -mixedsvvh [b | l | r][i]

  (optional) Facilitates using VHDL packages at the SystemVerilog-VHDL boundary of a mixed-language design. When you compile a VHDL package with **-mixedsvvh**, the package can be included in a SystemVerilog design as if it were defined in SystemVerilog itself.

  Executing **-mixedsvvh** without arguments compiles VHDL vectors in the following ways:

  - VHDL bit_vectors are treated as SystemVerilog bit vectors.
  - VHDL std_logic_vectors, std_ulogic_vectors, and vl_logic_vectors are treated as SystemVerilog logic vectors.

  b — treats all scalars and vectors in the package as SystemVerilog bit type

  l — treats all scalars and vectors in the package as SystemVerilog logic type

  r — treats all scalars and vectors in the package as SystemVerilog reg type

  i — ignores the range specified with VHDL integer types. Can be specified together with b, l, or r, spaces are not allowed between arguments.

- -modelsimini <ini_filepath>

  (optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable.

  <ini_filepath> — Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

- -no1164

  (optional) Causes the source files to be compiled without taking advantage of the built-in version of the IEEE **std_logic_1164** package. This will typically result in longer simulation times for VHDL programs that use variables and signals of type **std_logic**.

- -noaccel <package_name>

  (optional) Turns off acceleration of the specified package in the source code using that package.

  <package_name> — A VHDL package name.

- -nocasestaticerror

  (optional) Suppresses case statement static warnings. VHDL standards require that case statement alternative choices be static at compile time. However, some expressions 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

  (optional) Disables index and range checks. You can disable these individually using the **-noindexcheck** and **-norangecheck** arguments, respectively.

- -nodbgsym

  (optional) Disables the generation of the symbols debugging database in the compiled library.

  The symbols debugging database is the *.dbs* file in the compiled library that provides information to the GUI allowing you to view detailed information about design objects at the source level. Two major GUI features that use this database include source window annotation and textual dataflow.

  You should only specify this switch if you know that anyone using the library will not require this information for design analysis purposes.

- -noDeferSubpgmCheck

  (optional) Causes range and length violations detected within subprograms to be reported as errors (instead of as warnings). As an alternative to using this argument, you can set the NoDeferSubpgmCheck variable in the modelsim.ini file to a value of 1.

- -nofprangecheck

  (optional) Disables range checks on floating type values only.

- -noFunctionInline

  (optional) Turns off VHDL subprogram inlining for design units using a local copy of a VHDL package. This may be needed in case the local package has the same name as an MTI supplied package.

- -noindexcheck

  (optional) Disables checking on indexing expressions to determine whether indexes are within declared array bounds.

- -nologo

  (optional) Disables display of the startup banner.

- -nonstddriverinit

  (optional) Forces ModelSim to match pre-5.7c behavior in initializing drivers in a particular case. Prior to 5.7c, VHDL ports of mode out or inout could have incorrectly initialized drivers if the port did not have an explicit initialization value and the actual signal connected to the port had explicit initial values. Depending on a number of factors, ModelSim could incorrectly use the actual signal's initial value when initializing lower level drivers. Note that the argument does not cause all lower-level drivers to use the actual signal's initial value. It does this only in the specific cases where older versions used the actual signal's initial value.

- -noothersstaticerror

  (optional) Disables warnings that result from array aggregates with multiple choices having "others" clauses that are not locally static. If **-pedanticerrors** is specified, this switch is ignored.

- -norangecheck

  (optional) 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**. If you run a simulation with range checking disabled, any scalar values that are out of range are indicated by showing the value in the following format: ?(N) where N is the current value. Refer to “Range and Index Checking” for additional information. I

- -note <msg_number> [,<msg_number>, …]

  (optional) Changes the severity level of the specified message(s) to "note. Edit the note variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

  <msg_number> — A number identifying the message. Multiple message numbers are specified as a comma separated list.

- -novital

  (optional) Causes **vcom** to use VHDL code for VITAL procedures rather than the accelerated and optimized timing and primitive packages built into the simulator kernel. Allows breakpoints to be set in the VITAL behavior process and permits single stepping through the VITAL procedures to debug your model. Also all of the VITAL data can be viewed in the Locals or Objects windows.

- -novitalcheck

  (optional) Disables Vital level 1 and Vital level 0 checks defined in section 4 of the Vital-95 Spec (IEEE Std 1076.4-1995).

- -nowarn <category_number>

  (optional) Selectively disables a category of warning messages. Warnings may be disabled for all compiles via the Main window **Compile > Compile Options** menu command or the *modelsim.ini* file (Refer to modelsim.ini Variables).

  <category_number> — Specifies one or more numbers corresponding to the categories in Table 2-5 Multiple message categories are specified as a comma separated list.

Table 2-5. Warning Message Categories for vcom -nowarn

Category number	Description
1	unbound component
2	process without a wait statement
3	null range
4	no space in time literal
5	multiple drivers on unresolved signal
6	VITAL compliance checks (“VitalChecks” also works)
7	VITAL optimization messages
8	lint checks
9	signal value dependency at elaboration
10	VHDL-1993 constructs in VHDL-1987 code
13	constructs that coverage can't handle
14	locally static error deferred until simulation run

- -oldconfigvis

  (optional) Forces **vcom** to process visibility of VHDL component configurations consistent with prior releases. Default behavior is to comply with Language Reference Manual visibility rules. Refer to **vcom -lrmVHDLConfigVis** or the modelsim.ini variable OldVHDLConfigurationVisibility for more information.

- -pedanticerrors

  (optional) Forces display of an error message (rather than a warning) on a variety of conditions. Refer to “Enforcing Strict 1076 Compliance” for a complete list of these conditions. This argument overrides **-nocasestaticerror** and **-noothersstaticerror** (see above).

- -performdefaultbinding

  (optional) Enables default binding when it has been disabled via the RequireConfigForAllDefaultBinding option in the *modelsim.ini* file.

- -preserve

  (optional) Forces **vcom** to preserve the case of letters in object identifiers. Can also be enabled by setting the PreserveCase variable to 1 in the modelsim.ini file.

- -quiet

  (optional) Disables 'Loading' messages.

- -rangecheck

  (default) Enables run time range checking. Range checking can be disabled using the **-norangecheck** argument. Refer to "Range and Index Checking" for additional information.

- -refresh

  (optional) Regenerates a library image. By default, the work library is updated. To update a different library, use **-work <library_name>** with **-refresh** (for example, v**com -work <your_lib_name> -refresh**). If a dependency checking error occurs which prevents the refresh, use the **vcom -force_refresh** argument. Refer to the **vcom** Examples for more information. You may use a specific design name with **-refresh** to regenerate a library image for that design, but you may not use a file name.

- -s

  (optional) Instructs the compiler not to load the **standard** package. This argument should only be used if you are compiling the **standard** package itself.

- -separateConfigLibrary

  Allows the declaration of a VHDL configuration to occur in a different library than the entity being configured. Strict conformance to the VHDL standard (LRM) requires that they be in the same library. This argument must be used if optimization is disabled (-novopt).

- -skip abcep

  (optional) 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 switch.

- -source

  (optional) Displays the associated line of source code before each error message that is generated during compilation. By default, only the error message is displayed.

- -suppress <msg_number>[,<msg_number>,…]

  (optional) Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. You cannot suppress Fatal or Internal

messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

<msg_number> — A number identifying the message. Multiple message numbers are specified as a comma separated list.

- -time

  (optional) Reports the "wall clock time" **vcom** takes to compile the design. 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

  (optional) Returns the version of the compiler as used by the licensing tools.

- -vmake

  (optional) Generates a complete record of all command line data and files accessed during the compile of a design. This data is then used by the vmake command to generate a comprehensive makefile for recompiling the design library. By default, **vcom** stores compile data needed for the **-refresh** switch and ignores compile data not needed for **-refresh**. The **-vmake** switch forces inclusion of all file dependencies and command line data accessed during a compile, whether they contribute data to the initial compile or not. Executing this switch can increase compile time in addition to increasing the accuracy of the compile. See the **vmake** command for more information.

- -warning <msg_number>[,<msg_number>,…]

  (optional) Changes the severity level of the specified message(s) to "warning." Edit the warning variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

  <msg_number> — A number identifying the message. Multiple message numbers are specified as a comma separated list.

- -work <library_name>

  (optional) Maps a library to the logical library **work**. By default, the compiled design units are added to the **work** library. The specified pathname overrides the pathname specified for work in the project file.

  <library_name> — A logical name or pathname of a library.

- <filename>

  (required) 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 .Wildcards may be used, for example, **.vhd*.

  If you don't specify a filename, and you are using the GUI, a dialog box pops up allowing you to select the options and enter a filename.

Examples

- Compile the VHDL source code contained in the file *example.vhd*.

```
vcom example.vhd
```

- ModelSim supports designs that use elements conforming to the 1987, 1993, and 2002 standards. Compile the design units separately using the appropriate switches.

  ```
  vcom -87 o_units1.vhd o_units2.vhd
  vcom -93 n_unit91.vhd n_unit92.vhd
  ```

- When compiling source that uses the **numeric_std** package, this command turns off acceleration of the **numeric_std** package, located in the **ieee** library.

  ```
  vcom -noaccel numeric_std example.vhd
  ```

- Although it is not obvious, the = operator is overloaded in the **std_logic_1164** package. All enumeration data types in VHDL get an “implicit” definition for the = operator. So while there is no explicit = operator, there is an implicit one. This implicit declaration can be hidden by an explicit declaration of = in the same package (LRM Section 10.3). However, if another version of the = operator is declared in a different package than that containing the enumeration declaration, and both operators become visible through **use** clauses, neither can be used without explicit naming.

  ```
  vcom -explicit example.vhd
  ```

  To eliminate that inconvenience, the VCOM command has the **-explicit** option that allows the explicit = operator to hide the implicit one. Allowing the explicit declaration to hide the implicit declaration is what most VHDL users expect.

  ```
  ARITHMETIC."="(left, right)
  ```

- The **-work** option specifies **mylib** as the library to regenerate. **-refresh** rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of ModelSim (4.6 and later only).

  ```
  vcom -work mylib -refresh
  ```

vdel

This command deletes a design unit from a specified library.

This command provides additional information with the **-help** switch.

Syntax

vdel [-lib <library_path>] [-modelsimini <ini_filepath>] [-verbose]
{ -all | <primary> [<arch_name>]| -obj [object_info] | -dpiobj [object_info] }

Arguments

- -all

  (optional) Deletes an entire library.

Caution

You cannot recover libraries cannot once deleted. You are not prompted for confirmation.

- -dpiobj [<object_info]

  (optional) Delete auto-compiled DPI object files.

  <object_info> — Specifies the type of object to remove, as reported by the output of the vdir -obj command. This will take the form of either:

  <compiler> — a string identifying the compiler, such as **gcc-3.3.1**.

  <platform> — a string identifying the platform, such as **linux**.

  <platform-compiler> — a string identifying a compiler/platform pair, such as **linux_gcc-3.2.3**.

  all — Specifies that all objects should be removed, as reported by the output of the vdir -obj command.

- -lib <library_path>

  (optional) Specifies location of the library that holds the design unit to be deleted. By default, the design unit is deleted from the **work** library.

  <library_path> — A logical name or pathname of the library.

- -modelsimini <ini_filepath>

  (optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable.

  <ini_filepath> — Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

- -obj {<object_info>}

  (optional) removes directories containing DPI object files.

  <object_info> — Specifies the type of directory to remove, as reported by the output of the vdir -obj command. This will take the form of either:

<compiler> — a string identifying the compiler, such as **gcc-3.3.1**.

<platform> — a string identifying the platform, such as **linux**.

<platform-compiler> — a string identifying a compiler/platform pair, such as **linux_gcc-3.2.3**.

all — Specifies that all directories should be removed, as reported by the output of the vdir -obj command.

- <primary> [<arch_name>]

  (required unless **-all** is used) Specifies the entity, package, configuration, or module to be deleted.

  <arch_name> — Specifies the name of an architecture to be deleted. If omitted, all of the architectures for the specified entity are deleted. Invalid for a configuration or a package.

- -verbose

  (optional) Displays progress messages.

Examples

- Delete the **work** library.

  ```
  vdel -all
  ```

- Delete the **synopsys** library.

  ```
  vdel -lib synopsys -all
  ```

- Delete the entity named **xor** and all its architectures from the **work** library.

  ```
  vdel xor
  ```

- Delete the architecture named **behavior** of the entity **xor** from the **work** library.

  ```
  vdel xor behavior
  ```

- Delete the package named **base** from the **work** library.

  ```
  vdel base
  ```

vdir

This command lists the contents of a design library and checks the compatibility of a vendor library. If **vdir** cannot read a vendor-supplied library, the library may not be compatible with ModelSim.

This command provides additional information with the **-help** switch.

Syntax

vdir [-l | [-prop <prop>]] [-r] [-all | [-lib <library_name>]] [<design_unit>]
[-modelsimini <ini_filepath>]

Arguments

- -all

  (optional) Lists the contents of all libraries listed in the Library section of the active *modelsim.ini* file. Refer to modelsim.ini Variables for more information.

- <design_unit>

  (optional) Indicates the design unit to search for within the specified library. If the design unit is a VHDL entity, its architectures are listed. By default all entities, configurations, modules, packages, and optimized design units in the specified library are listed.

- -l

  (optional) Prints the version of **vcom/vlog** with which each design unit was compiled, plus any compilation options used. Also prints the object-code version number that indicates which versions of **vcom/vlog** and ModelSim are compatible.

- -lib <library_name>

  (optional) Specifies the logical name or the pathname of a library to be listed. By default, the contents of the **work** library are listed.

  <library_name> — A logical name or pathname of a library.

- -modelsimini <ini_filepath>

  (optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable.

  <ini_filepath> — Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

- -prop <prop>

  (optional) Reports on a specified design unit property.

<prop> — Specifies a Design Unit Property, as listed in Table 2-6. If you do not specify a value for <prop>, an error message is displayed.

Table 2-6. Design Unit Properties

Value of <prop>	Description
archcfg	configuration for arch
body	needs a body
default	default options
dir	source directory
dpnd	depends on
entcfg	configuration for entity
fulloptions	Full compile options
inline	module inlined
lock	lock/unlock status
lrm	language standard
mtime	source modified time
name	short name
opcode	opcode format
options	compile options
pdu	preoptimized design unit
root	optimized Verilog design root
src	source file
top	top level model
ver	version string
vlogv	Verilog version

- -r

  (optional) Prints architecture information for each entity in the output.

Examples

- List the architectures associated with the module named **and2** that reside in the default library work.

  vdir -l and2

```
# Library vendor : Model Technology
# Maximum unnamed designs : 3
```

```
# MODULE and2
#     Verilog version: <XO@d;_mSdz@12Fz9b]_Z3
#     Version string: 3EdggZ>V3z51fE;>K[51?2
#     Source directory: C:\examples\dataflow_verilog
#     Source modified time: Tue Apr 28 22:48:56 2009
#     HDL source file: gates.v
#     Source file: gates.v
#     Start location: gates.v:18
#     Opcode format: 10.1a; VLOG SE Object version 51
#     Optimized Verilog design root: 1
#     VHDL language standard: 1
#     Compile options: -L mtiAvm -L mtiOvm -L mtiUvm -L mtiUPF
#     Debug Symbol file exists
```

vencrypt

This command encrypts Verilog and SystemVerilog code contained within encryption envelopes. The code is not pre-processed before encryption, so macros and other `directives are unchanged. This allows IP vendors to deliver encrypted IP with undefined macros and `directives.

Upon execution of this command, the filename extension will be changed to *.vp* for Verilog files (*.v* files) and *.svp* for SystemVerilog files (*.sv* files).

If the vencrypt utility processes the file (or files) and does not find any encryption directives it reprocesses the file using the following default encryption:

```
`pragma protect data_method = "aes128-cbc"
`pragma protect key_keyowner = "MTI"
`pragma protect key_keyname = "MGC-DVT-MTI"
`pragma protect key_method = "rsa"
`pragma protect key_block encoding = (enctype = "base64")
`pragma protect begin
```

The vencrypt command must be followed by a compile command – such as vlog – for the design to be compiled.

This command provides additional information with the **-help** switch.

Syntax

vencrypt **<filename>** [-d <dirname>] [-e <extension>] [-f <filename>] [-h <filename>] [-l <filename>] [-o <filename>] [-p <prefix>] [-quiet]

- <filename>

  (required) Specifies the name of the Verilog source code file to encrypt. One filename is required. Multiple filenames can be entered separated by spaces. Wildcards can be used. Default encryption pragmas will be used, as described above, if no encryption directives are found during processing.

- -d <dirname>

  (optional) Specifies where to save encrypted Verilog files. If no directory is specified, current working directory will be used.

  <dirname> — Specifies the directory to contain the encrypted Verilog or SystemVerilog files. The original file extension (*.v* for Verilog and *.sv* for SystemVerilog) will be preserved.

- -e <extension>

  (optional) Specifies a filename extension.

  <extension> — Any alpha-numeric string.

- -f <filename>

  (optional) Specifies a file with more command line arguments. Allows complex arguments to be reused without retyping. Nesting of **-f** options is allowed.

Refer to the section "Argument Files" for more information.

<filename> — Specifies the name of a file containing command line arguments.

- -h <filename>

  (optional) Concatenates header information into all design files listed with <filename>. Allows the user to pass a large number of files to the vencrypt utility that do not contain the **`pragma protect** or **`protect** information about how to encrypt the file. Saves the user from editing hundreds of files to add in the same `pragma protect to every file.

  <filename> — Specifies an existing file.

- -l <filename>

  (optional) Redirects log output to the file designated by <filename>.

  <filename> — Specifies a file for saving output.

- -o <filename>

  (optional) Combines all encrypted output into a single file.

  <filename> — Specifies a file for saving output.

- -p <prefix>

  (optional) Prepends file names with a prefix.

  <prefix> — Any alpha-numeric string.

- -quiet

  (optional) Disables encryption messages.

Example

- Insert header information into all design files listed.

  vencrypt -h encrypt_head top.v cache.v gates.v memory.v

  The *encrypt_head* file may look like the following:

```
`pragma protect data_method = "aes128-cbc"
`pragma protect author = "IP Provider"
`pragma protect key_keyowner = "MTI", key_method = "rsa"
`pragma protect key_keyname = "MGC-DVT-MTI"
`pragma protect begin
```

  There is no **`pragma protect end** expression in the header file, just the header block that starts the encryption. The **`pragma protect end** expression is implied by the end of the file. For more detailed examples, see "Protecting Your Source Code" in the User's Manual.

Related Topics

- vhencrypt
- "Protecting Your Source Code" in the User's Manual

verror

This command prints a detailed description about a message number. It may also point to additional documentation related to the error.

Syntax

verror [-fmt | -tag | -fmt -tag | -full] <msgNum> ...

verror [-fmt | -tag | -fmt -tag | -full] [-tool <tool>] -all

verror [-tool <tool>] -ranges

verror [-kind <tool>] {-pedanticerrors -permissive}

Arguments

- -fmt | -tag | -fmt -tag | -full

  (optional) Specifies the type and amount of information to return.

  -fmt — returns the format string used in the error message.

  -tag — returns the tag associated with the error message.

  -full — returns the format string, tag, and complete text associated with the error message.

- -tool <tool> -all

  (required when not specifying **<msgNum>** or **-ranges**) Returns information about all messages associated with a specified tool, where <tool> can be one of the following:

common	vcom	vcom-vlog
vlog	vsim	vsim-vish
wlf	vsim-sccom	sccom
vsim-systemc	ucdb	vsim-vlog
pseudo_synth		

- -kind <tool>

  (optional) Specifies filtering for messages according to either or both of the following:

  -pedanticerrors — display messages that are reported as errors due to adhering to a more strict interpretation of the LRM.

  -permissive — display messages reported as warnings that would be displayed as errors if you use vsim -pedanticerrors.

  where <tool> can be any of the values allowed for the -tool argument (above).

- <msgNum>

  (required when not specifying **-all** or **-ranges**) Specifies the message number(s) you would like more information about. You can find the message number in messages of the format:

** <Level>: ([<Tool>-[<Group>-]]<MsgNum>) <FormattedMsg>

You can specify <msgNum> any number of times for one verror command in a space-separated list.

Optionally, you can specify the toolname prior to the message number, similar to how it appears in an error message. For example:

```
verror vsim-5003
```

- -ranges

  (required when not specifying **<msgNum>** or **-all**) Prints the numeric ranges of error message numbers, organized by tool.

Example

- If you receive the following message in the transcript:

  ```
  ** Error (vsim-3061) foo.v(22): Too many Verilog port connections.
  ```

  and you would like more information about this message, you would type:

  verror 3061

  and receive the following output:

  ```
  Message # 3061:
  Too many Verilog ports were specified in a mixed VHDL/Verilog
  instantiation. Verify that the correct VHDL/Verilog connection is
  being made and that the number of ports matches.
  [DOC: ModelSim User's Manual - Mixed VHDL and Verilog Designs
  Chapter]
  ```

vgencomp

Once a Verilog module is compiled into a library, you can use this command to write its equivalent VHDL component declaration to standard output.

Optional switches allow you to generate bit or vl_logic port types; std_logic port types are generated by default.

This command provides additional information with the **-help** switch.

Syntax

vgencomp [-lib <library_name>] [-b] [-modelsimini <ini_filepath>] [-s] [-v] **<module_name>**

Arguments

- -lib <library_name>

  (optional) Specifies the working library where the default is to use the **work** library.

  <library_name> — Specifies the path and name of the working library.

- -b

  (optional) Causes **vgencomp** to generate bit port types.

- -modelsimini <ini_filepath>

  (optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable.

  <ini_filepath> — Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

- -s

  (optional) Used for the explicit declaration of default std_logic port types.

- -v

  (optional) Causes **vgencomp** to generate vl_logic port types.

- <module_name>

  (required) Specifies the name of the Verilog module to be accessed.

Examples

- This example uses a Verilog module that is compiled into the **work** library. The module begins as Verilog source code:

```
module top(i1, o1, o2, io1);
   parameter width = 8;
   parameter delay = 4.5;
   parameter filename = "file.in";

   input i1;
   output [7:0] o1;
   output [4:7] o2;
   inout [width-1:0] io1;
endmodule
```

After compiling, **vgencomp** is invoked on the compiled module:

vgencomp top

and writes the following to stdout:

```
component top
    generic(
      width           : integer := 8;
      delay           : real    := 4.500000;
      filename        : string  := "file.in"
   );
   port(
      i1              : in    std_logic;
      o1              : out   std_logic_vector(7 downto 0);
      o2              : out   std_logic_vector(4 to 7);
      io1             : inout std_logic_vector
   );
end component;
```

vhencrypt

This command encrypts VHDL code contained within encryption envelopes. The code is not compiled before encryption, so dependent packages and design units do not have to exist before encryption.

Upon execution of this command, the *.vhd* filename extension is changed to *.vhdp* and the *.vhdl* filename extension is changed to *.vhdlp*.

If the vhencrypt utility does not find any encryption directives, no output file is produced.

The vhencrypt command must be followed by a compile command – such as vcom – for the design to be compiled.

This command provides additional information with the **-help** switch.

Syntax

vhencrypt **<filename>** [-d <dirname>] [-e <extension>] [-f <filename>] [-h <filename>] [-l <filename>] [-o <filename>] [-p <prefix>] [-quiet]

- <filename>

  (required) Specifies the name of the VHDL source code file to encrypt. One filename is required. Multiple filenames can be entered separated by spaces. Wildcards can be used.

- -d <dirname>

  (optional) Specifies where to save encrypted VHDL files. If no directory is specified, the current working directory will be used.

  <dirname> — Specifies the directory to contain the encrypted VHDL files. The original file extension (*.vhd* or *.vhdl*) will be preserved.

- -e <extension>

  (optional) Specifies a filename extension to be applied to the encrypted file.

  <extension> — Any alpha-numeric string.

- -f <filename>

  (optional) Specifies a file with more command line arguments. Allows complex arguments to be reused without retyping. Nesting of **-f** options is allowed.

  Refer to the section "Argument Files" for more information.

  <filename> — Specifies the name of a file containing command line arguments.

- -h <filename>

  (optional) Concatenates header information into all design files listed with <filename>. Allows the user to pass a large number of files to the vhencrypt utility that do not contain the encryption information (between the **`protect** and **`protect end** directives) about how to encrypt the file. Saves the user from editing hundreds of files to add the same encryption information into every file.

  <filename> — Specifies an existing file.

- -l <filename>

  (optional) Redirects log output to the file designated by <filename>.

  <filename> — Specifies a file for saving output.
- -o <filename>

  (optional) Combines all encrypted output into a single file.

  <filename> — Specifies a file for saving output.
- -p <prefix>

  (optional) Prepends encrypted file names with a prefix.

  <prefix> — Any alpha-numeric string.
- -quiet

  (optional) Disables encryption messages.

Related Topics

- vencrypt
- "Protecting Your Source Code" in the User's Manual

view

This command opens the specified window. If you specify this command without arguments it returns a list of all open windows in the current layout.

To remove a window, use the noview command.

The **view** command with one or more options and no window names specified applies the options to the currently open windows. See examples for additional details.

Syntax

view **<window_type>…**[-aliases][-names] [-title {New Window Title}]
[-undock {[-icon] [-height <n>] [-width <n>] [-x <n>] [-y <n>]} | -dock]

Arguments

- <window_type>…

  (required) Specifies the window type to view. You do not need to type the full type name (see the examples below); implicit wildcards are accepted; multiple window types are accepted. Available window types are:

assertions	atv	browser	calltree
canalysis	capacity	classgraph	classtree
covergroups	dataflow	details	duranked
exclusions	fcovers	files	fsmcoverage
fsmlist	fsmview	instance	library
list	locals	memdata	memory
msgviewer	objects	process	profiledetails
project	ranked	runmgr	schematic
source	stackview	structural	structure
tracker	transaction	transcript	watch
wave			

  Not all windows are available with all variants (ModelSim SE, ModelSim PE, Questa SV/AFV, and so on)

- -aliases

  (optional) Returns a list of <window_type> aliases.

- -height <n>

  (optional) Specifies the window height in pixels. Can only be used with the **-undock** switch.

  <n> — Any non-negative integer.

- -icon

  (optional) Toggles the view between window and icon. Can only be used with the **-undock** switch.

- -names

  (optional) Returns a list of valid <window_type> arguments.

- -title {New Window Title}

  (optional) Specifies the window title of the designated window.

  {New Window Title} — Any string. Curly braces are needed for a string containing spaces. Double quotes (" ") can be used in place of braces, for example "New Window Title."

- -dock

  (optional) Docks the specified standalone window into the Main window.

- -undock

  (optional) Opens the specified window as a standalone window, undocked from the Main window.

- -width <n>

  (optional) Specifies the window width in pixels. Can only be used with the **-undock** switch.

  <n> — Any non-negative integer.

- -x <n>

  (optional) Specifies the window upper-left-hand x-coordinate in pixels. Can only be used with the **-undock** switch.

  <n> — Any non-negative integer.

- -y <n>

  (optional) Specifies the window upper-left-hand y-coordinate in pixels. Can only be used with the **-undock** switch.

  <n> — Any non-negative integer.

Examples

- Undock the Wave window from the Main window and makes it a standalone window.

  ```
  view -undock wave
  ```

- Display an undocked Processes window in the upper left-hand corner of the monitor with a window size of 300 pixels, square.

  ```
  view process -undock -x 0 -y 0 -width 300 -height 300
  ```

- Display the Watch and Wave windows.

```
view w
```

- Display the Objects and Processes windows.

```
view ob pr
```

- Open a new Wave window with My Wave Window as its title.

```
view -title {My Wave Window} wave
```

Related Topics

- noview

virtual count

This command reports the number of currently defined virtuals that were not read in using a macro file.

Syntax

virtual count [-kind {implicits | explicits}] [-unsaved]

Arguments

- -kind {implicits | explicits}

  (optional) Reports only a subset of virtuals.

  implicits — virtual signals created internally by the product.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

- -unsaved

  (optional) Reports the count of only those virtuals that have not been saved to a macro file.

Related Topics

- virtual define
- virtual save
- virtual show
- Virtual Objects

virtual define

This command prints to the transcript the definition of the virtual signals, functions, or regions in the form of a command that can be used to re-create the object.

Syntax

virtual define [-kind {implicits | explicits>] <pathname>

Arguments

- -kind {implicits | explicits}

  (optional) Transcripts only a subset of virtuals.

  implicits — virtual signals created internally by the tool.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

- <pathname>

  (required) Specifies the path to the virtual(s) for which you want definitions, where wildcards are allowed.

Example

- Show the definitions of all the virtuals you have explicitly created.

```
virtual define -kind explicits *
```

Related Topics

- virtual describe
- virtual show
- Virtual Objects

virtual delete

This command removes the matching virtuals.

Syntax

virtual delete [-kind {implicits | explicits}] <pathname>

Arguments

- -kind {implicits | explicits}

  (optional) Removes only a subset of virtuals.

  implicits — virtual signals created internally by the product.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

- <pathname>

  (required) Specifies the path to the virtual(s) you want to delete, where wildcards are allowed.

Example

- Delete all of the virtuals you have explicitly created.

```
virtual delete -kind explicits *
```

Related Topics

- virtual signal
- virtual function
- Virtual Objects

virtual describe

This command prints to the transcript a complete description of the data type of one or more virtual signals.

Similar to the existing **describe** command.

Syntax

virtual describe [-kind {implicits | explicits}] <pathname>

Arguments

- -kind {implicits | explicits}

  (optional) Transcripts only a subset of virtuals.

  implicits — virtual signals created internally by the product.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

- <pathname>

  (required) Specifies the path to the virtual(s) for which you want descriptions, where wildcards are allowed.

Example

- Describe the data type of all virtuals you have explicitly created.

```
virtual describe -kind explicits *
```

Related Topics

- virtual define
- virtual show
- Virtual Objects

virtual expand

This command prints to the transcript a list of all the non-virtual objects contained in the specified virtual signal(s).

You can use this to create a list of arguments for a command that does not accept or understand virtual signals.

Syntax

virtual expand [-base] <pathname> ...

Arguments

- -base

  (optional) Outputs the root signal parent in place of a subelement. For example:

  ```
  vcd add [virtual expand -base myVirtualSignal]
  ```

  the resulting command after substitution would be:

  ```
  vcd add signala signalb signalc
  ```

- <pathname>

  (required) Specifies the path to the signals and virtual signals to expand, where wildcards are allowed and you can specify any number of paths.

Examples

- Add the elements of a virtual signal to the VCD file.

  In the Tcl language, the square brackets specify that the enclosed command should be executed first ("virtual expand …"), then the result substituted into the surrounding command.

  ```
  vcd add [virtual expand myVirtualSignal]
  ```

  Therefore, if *myVirtualSignal* is a concatenation of *signala*, *signalb.rec1* and *signalc(5 downto 3)*, the resulting command after substitution would be:

  ```
  vcd add signala signalb.rec1 {signalc(5 downto 3)}
  ```

  The slice of *signalc* is enclosed in curly braces, because it contains spaces.

Related Topics

- virtual signal
- Virtual Objects

virtual function

This command creates a new signal, known only by the GUI (not the kernel), that consists of logical operations on existing signals and simulation time, as described in **<expressionString>**.

It cannot handle bit selects and slices of Verilog registers. Please see *Syntax and Conventions* for more details on syntax.

If the virtual function references more than a single scalar signal, it will display as an expandable object in the Wave and Objects windows. The children correspond to the inputs of the virtual function. This allows the function to be "expanded" in the Wave window to see the values of each of the input waveforms, which could be useful when using virtual functions to compare two signal values.

Virtual functions can also be used to gate the List window display.

Note

The virtual function and virtual signal commands are interchangeable. The product will keep track of whether you've created a signal or a function with the commands and maintain them appropriately. We document both commands because the virtual save, virtual describe, and virtual define commands will reference your virtual objects using the correct command.

Syntax

virtual function [-env <path>] [-install <path>] [-delay <time>] {<expressionString>} <name>

Arguments

Arguments for **virtual function** are the same as those for **virtual signal**, except for the contents of the expression string.

- -env <path>

  (optional) Specifies a hierarchical context for the signal names in **<expressionString>** so they don't all have to be full paths.

  <path> — Specifies a relative path to the signal(s). On Windows systems the path separator should be a forward slash (/).

- -install <path>

  (optional) Causes the newly-created signal to become a child of the specified region. If **-install** is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in **<expressionString>**. If the expression references more than one WLF file (dataset), the virtual signal will automatically be placed in region virtuals:/Functions.

  <path> — Specifies a relative path to the signal(s). On Windows systems the path separator should be a forward slash (/).

- -delay <time> <unit>

  (optional) Specifies a value by which the virtual function will be delayed. You can use negative values to look forward in time. Refer to the examples below for more details.

  <time> — Specified as an integer or decimal number. Current simulation units are the default unless specifying **<unit>**.

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

- {<expressionString>}

  (required) A text string expression, enclosed in curly braces ({ }) using the GUI_expression_format.

- <name>

  (required) The name you define for the virtual signal.

  Case is ignored unless installed in a Verilog region.

  Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation.

  If using VHDL extended identifier notation, **<name>** needs to be quoted with double quotes (" ") or with curly braces ({ }).

Examples

- Create a signal */chip/section1/clk_n* that is the inverse of */chip/section1/clk*.

  ```
  virtual function { not /chip/section1/clk } clk_n
  ```

- Create a std_logic_vector equivalent of a Verilog register *rega* and installs it as */chip/rega_slv*.

  ```
  virtual function -install /chip { (std_logic_vector) chip.vlog.rega
  } rega_slv
  ```

- Create a boolean signal */chip/addr_eq_fab* that is true when */chip/addr[11:0]* is equal to hex "fab", and false otherwise. It is acceptable to mix VHDL signal path notation with Verilog part-select notation.

  ```
  virtual function { /chip/addr[11:0] == 0xfab } addr_eq_fab
  ```

- Create a signal that is high only during times when signal */chip/siga* of the gate-level version of the design does not match */chip/siga* of the rtl version of the design. Because there is no common design region for the inputs to the expression, *siga_diff* is installed in region *virtuals:/Functions*. The virtual function *siga_diff* can be added to the Wave window, and when expanded will show the two original signals that are being compared.

```
virtual function { gate:/chip/siga XOR rtl:/chip/siga } siga_diff
```

- Create a virtual signal consisting of the logical "AND" function of */top/signalA* with */top/signalB*, and delays it by 10 ns.

```
virtual function -delay {10 ns} {/top/signalA AND /top/signalB}
myDelayAandB
```

- Create a one-bit signal *outbus_diff* which is non-zero during times when any bit of */chip/outbus* in the gate-level version doesn't match the corresponding bit in the rtl version.

  This expression uses the "OR-reduction" operator, which takes the logical OR of all the bits of the vector argument.

```
virtual function { | (gate:/chip/outbus XOR rtl:/chip/outbus) }
outbus_diff
```

Commands fully compatible with virtual functions

add log and log | delete | describe
examine | find | restart
searchlog | show

Commands not compatible with virtual functions

drivers | force | noforce
vcd add | when

Related Topics

- virtual count
- virtual describe
- virtual log
- virtual region
- virtual signal
- virtual define
- virtual expand
- virtual nohide
- virtual save
- virtual type
- virtual delete
- virtual hide
- virtual nolog
- virtual show
- Virtual Objects

virtual hide

This command causes the specified real or virtual signals to not be displayed in the Objects window. This is used when you want to replace an expanded bus with a user-defined bus.

You make the signals reappear using the **virtual nohide** command.

Syntax

virtual hide { [-kind {implicits | explicits}] | [-region <path>]} <pattern>

Arguments

- -kind {implicits | explicits}

  (optional) Hides only a subset of virtuals.

  implicits — virtual signals created internally by the tool.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

- -region <path>

  (optional) Specifies a region of design space in which to look for the signal names.

  <path> — Specifies an absolute or relative path to the signal(s). On Windows systems the path separator should be a forward slash (/).

- <pattern>

  (required) Indicates which signal names or wildcard patterns should be used in finding the signals to hide, where wildcards are allowed and you can specify any number of names or patterns.

Related Topics

- virtual nohide
- Virtual Objects

virtual log

This command causes the simulation-mode dependent signals of the specified virtual signals to be logged by the kernel.

If wildcard patterns are used, it will also log any normal signals found, unless the **-only** option is used. You unlog the signals using the **virtual nolog** command.

Syntax

virtual log [-kind {implicits | explicits}] | [-region <path>] [-recursive] [-only] [-in] [-out] [-inout] [-internal] [-ports] <pattern>

Arguments

- -kind {implicits | explicits}

  (optional) Logs only a subset of virtuals.

  implicits — virtual signals created internally by the tool.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

- -region <path>

  (optional) Specifies a region of design space in which to look for signals to log.

  <path> — Specifies an absolute or relative path to the signal(s). On Windows systems the path separator should be a forward slash (/).

- -recursive

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

- -only

  (optional) Specify that only virtual signals (as opposed to all signals) found by a <pattern> containing a wildcard should be logged.

- -in

  (optional) Specifies that the kernel log data for ports of mode IN whose names match the specification.

- -out

  (optional) Specifies that the kernel log data for ports of mode OUT whose names match the specification.

- -inout

  (optional) Specifies that the kernel log data for ports of mode INOUT whose names match the specification.

- -internal

  (optional) Specifies that the kernel log data for internal (non-port) objects whose names match the specification.

- -ports

  (optional) Specifies that the kernel log data for all ports. Optional.

- <pattern>

  (required) Indicates which signal names or wildcard patterns should be used in finding the signals to log, where you can specify any number of names or wildcard patterns.

Related Topics

- virtual nolog
- Virtual Objects

virtual nohide

This command reverses the effect of a **virtual hide** command, causing the specified real or virtual signals to reappear the Objects window.

Syntax

virtual nohide { [-kind {implicits | explicits}] | [-region <path>] } <pattern>

Arguments

- -kind {implicits | explicits}

  (optional) Unhides only a subset of virtuals.

  implicits — virtual signals created internally by the tool.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

- -region <path>

  (optional) Specifies a region of design space in which to look for the signal names.

  <path> — Specifies an absolute or relative path to the signal(s). On Windows systems the path separator should be a forward slash (/).

- <pattern>

  (required) Indicates which signal names or wildcard patterns should be used in finding the signals to hide, where wildcards are allowed and you can specify any number of names or patterns.

Related Topics

- virtual hide
- Virtual Objects

virtual nolog

This command reverses the effect of a **virtual log** command. It causes the simulation-dependent signals of the specified virtual signals to be excluded ("unlogged") by the kernel.

If wildcard patterns are used, it will also unlog any normal signals found, unless the **-only** option is used.

Syntax

virtual nolog { [-kind {implicits | explicits}] | [-region <path>]} [-recursive] [-only] [-in] [-out] [-inout] [-internal] [-ports] <pattern>

Arguments

- -kind {implicits | explicits}

  (optional) Excludes only a subset of virtuals.

  implicits — virtual signals created internally by the tool.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

- -region <path>

  (optional) Specifies a region of design space in which to look for signals to unlog.

  <path> — Specifies an absolute or relative path to the signal(s). On Windows systems the path separator should be a forward slash (/).

- -recursive

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

- -only

  (optional) Specify that only virtual signals (as opposed to all signals) found by a <pattern> containing a wildcard should be unlogged.

- -in

  (optional) Specifies that the kernel exclude data for ports of mode IN whose names match the specification.

- -out

  (optional) Specifies that the kernel exclude data for ports of mode OUT whose names match the specification.

- -inout

  (optional) Specifies that the kernel exclude data for ports of mode INOUT whose names match the specification.

- -internal

  (optional) Specifies that the kernel exclude data for internal (non-port) objects whose names match the specification.

- -ports

  (optional) Specifies that the kernel exclude data for all ports.

- <pattern>

  (required) Indicates which signal names or wildcard patterns should be used in finding the signals to unlog, where wildcards are allowed and you can specify any number of names or patterns.

Related Topics

- virtual log
- Virtual Objects

virtual region

This command creates a new user-defined design hierarchy region.

Note

Virtual regions cannot be used in the when command.

Syntax

virtual region <parentPath> <regionName>

Arguments

- <parentPath>

  (required) The full path to the region that will become the parent of the new region.

- <regionName>

  (required) The name you want for the new region.

Related Topics

- virtual function
- virtual signal
- Virtual Objects

virtual save

This command saves the definitions of virtuals to a file named virtual.do in the current directory.

Syntax

virtual save [-kind {implicits | explicits}] [-append] [<filename>]

Arguments

- -kind {implicits | explicits}

  (optional) Saves only a subset of virtuals.

  implicits — virtual signals created internally by the tool.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

- -append

  (optional) Specifies to save **only** virtuals that are not already saved or weren't read in from a macro file. These unsaved virtuals are then appended to the specified or default file. Optional.

- <filename>

  (optional) The name of the file containing the definitions. If you don't specify **<filename>**, the default virtual filename (*virtuals.do*) will be used. You can specify a different default in the *pref.tcl* file.

Related Topics

- virtual count
- Virtual Objects

virtual show

This command lists the full path names of all explicitly defined virtuals.

Syntax

- virtual show [-kind {implicits | explicits}]

Arguments

- -kind {implicits | explicits}

  (optional) Lists only a subset of virtuals.

  implicits — virtual signals created internally by the tool.

  explicits — virtual signals explicitly created by a user, such as with the virtual signal command.

  Unique abbreviations are accepted.

Related Topics

- virtual define
- virtual describe
- Virtual Objects

virtual signal

This command creates a new signal, known only by the GUI (not the kernel), that consists of concatenations of signals and subelements as specified in **<expressionString>**.

It cannot handle bit selects and slices of Verilog registers. Please see Concatenation of Signals or Subelements for more details on syntax.

Note

The virtual function and virtual signal commands are interchangeable. The product will keep track of whether you've created a signal or a function with the commands and maintain them appropriately. We document both commands because the virtual save, virtual describe, and virtual define commands will reference your virtual objects using the correct command.

Syntax

virtual signal [-env <path>] [-install <path>] [-delay <time>] {<expressionString>} <name>

Arguments

- -env <path>

  (optional) Specifies a hierarchical context for the signal names in **<expressionString>** so they don't all have to be full paths.

  <path> — Specifies a relative path to the signal(s). On Windows systems the path separator should be a forward slash (/).

- -install <path>

  (optional) Causes the newly-created signal to become a child of the specified region. If **-install** is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in **<expressionString>**. If the expression references more than one WLF file (dataset), the virtual signal will automatically be placed in region virtuals:/Signals.

  <path> — Specifies a relative path to the signal(s). On Windows systems the path separator should be a forward slash (/).

- -delay <time> <unit>

  (optional) Specifies a value by which the virtual function will be delayed. You can use negative values to look forward in time. Refer to the examples below for more details.

  <time> — Specified as an integer or decimal number. Current simulation units are the default unless specifying **<unit>**.

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

- {<expressionString>}

  (required) A text string expression, enclosed in curly braces ({ }) using the GUI_expression_format.

- <name>

  (required) The name you define for the virtual signal.

  Case is ignored unless installed in a Verilog region.

  Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation.

  If using VHDL extended identifier notation, **<name>** needs to be quoted with double quotes (" ") or with curly braces ({ }).

Examples

- Reconstruct a bus *sim:/chip/alu/a(4 downto 0)*, using VHDL notation, assuming that *a_ii* are all scalars of the same type.

```
virtual signal -env sim:/chip/alu { (concat_range (4 downto 0))(a_04
& a_03 & a_02 & a_01 & a_00) } a
```

- Reconstruct a bus *sim:chip.alu.a[4:0]*, using Verilog notation. Note that the concatenation notation starts with "&{" rather than "{".

```
virtual signal -env sim:chip.alu
{ (concat_range [4:0])&{a_04, a_03, a_02, a_01, a_00} } a
```

- Create a signal *sim:/testbench/stuff* which is a record type with three fields corresponding to the three specified signals. The example assumes */chipa/mode* is of type integer, */chipa/alu/a* is of type std_logic_vector, and */chipa/decode/inst* is a user-defined enumeration.

```
virtual signal -install sim:/testbench
{ /chipa/alu/a(19 downto 13) & /chipa/decode/inst & /chipa/mode }
stuff
```

- Create a virtual signal that is the same as */top/signalA* except it is delayed by 10 ps.

```
virtual signal -delay {10 ps} {/top/signalA} myDelayedSignalA
```

- Create a three-bit signal, *chip.address_mode*, as an alias to the specified bits.

```
virtual signal { chip.instruction[23:21] } address_mode
```

- Concatenate signals *a*, *b*, and *c* with the literal constant ‘000’.

```
virtual signal {a & b & c & 3'b000} myextendedbus
```

- Add three missing bits to the bus *num*, creates a virtual signal *fullbus*, and then adds that signal to the Wave window.

```
virtual signal {num & "000"} fullbus
add wave -unsigned fullbus
```

- Reconstruct a bus that was fragmented by synthesis and is missing the lower three bits. Note that you would have to type in the actual bit names (i.e. num28, num27, and so on) represented by the … in the syntax above.

```
virtual signal { num31 & num30 & num29 & ... & num4 & num3 & "000" }
fullbus
add wave -unsigned fullbus
```

- Create a two-bit signal (with an enumerated type) based on the results of the subexpressions. For example, if *aold* equals *anew*, then the first bit is true (1). Alternatively, if *bold* does not equal *bnew*, the second bit is false (0). Each subexpression is evaluated independently.

```
virtual signal {(aold == anew) & (bold == bnew)} myequalityvector
```

- Create signal *newbus* that is a concatenation of bus1 (bit-reversed) and bus2[7:4] (bit-reversed). Assuming bus1 has indices running 7 downto 0, the result will be newbus[11:0] with the upper 8 bits being bus1[0:7] and the lower 4 bits being bus2[4:7]. See Concatenation Directives for further details.

```
virtual signal {(concat_reverse)(bus1 & bus2[7:4])} newbus
```

Commands fully compatible with virtual signals

add list	add log or log	add wave
delete	describe	examine
find	force and noforce	restart
searchlog	show	

Commands compatible with virtual signals using [virtual expand <signal>]

drivers	vcd add

Commands not currently compatible with virtual signals

when

Related Topics

- virtual count
- virtual define
- virtual delete
- virtual describe
- virtual expand
- virtual hide
- virtual log
- virtual nohide
- virtual nolog
- virtual region
- virtual save
- virtual show
- virtual function
- virtual type
- Virtual Objects

virtual type

This command creates a new enumerated type known only by the GUI, not the kernel. Virtual types are used to convert signal values to character strings. The command works with signed integer values up to 64 bits.

Virtual types cannot be used in the when command.

Note

If you are using SystemVerilog, you can also convert signal values to character strings using associative arrays in your code. See the SystemVerilog LRM for more information.

Syntax

virtual type -delete <name> | {<list_of_strings>} <name>

Arguments

- -delete <name>

  (Required if not defining a type.) Deletes a previously defined virtual type.

  <name> — The name you gave the virtual type when you originally defined it. .

- {<list_of_strings>}

  (Required if **-delete** is not used.) A list of values and their associated character strings. 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.

  If the string contains spaces the string must be enclosed in quotation marks ("") If they contain special characters square brackets, curly braces, backslashes…), they need to be quoted within curly braces.

  See the examples below for further syntax.

- <name>

  (Required if **-delete** is not used.) The user-defined name of the virtual type. Case is not ignored. Use alpha, numeric, and underscore characters only, unless you are using VHDL extended identifier notation. If using VHDL extended identifier notation, **<name>** needs to be quoted with double quotes (" ") or with curly braces ({ }).

Examples

- Using positional notation, associates each string with an enumeration index, starting at zero and increasing by one in the positive direction.When *myConvertedSignal* is displayed in the Wave, List, or Objects window, the string "state0" will appear when *mysignal* == 0, "state1" when *mysignal* == 1, "state2" when *mysignal* == 2, and so on.

```
virtual type {state0 state1 state2 state3} mystateType
virtual function {(mystateType)mysignal} myConvertedSignal
add wave myConvertedSignal
```

- Use sparse mapping of bus values to alphanumeric strings for an 8-bit, one-hot encoding. It shows the variety of syntax that can be used for values. The value "default" has special meaning and corresponds to any value not explicitly specified.

```
virtual type {{0 NULL_STATE} {1 st1} {2 st2} {0x04 st3} {16'h08 st4} \
             {'h10 st5} {16#20 st6} {0b01000000 st7} {0x80 st8} \
             {default BAD_STATE}} myMappedType
virtual function {(myMappedType)mybus} myConvertedBus
add wave myConvertedBus
```

- Delete the virtual type "mystateType".

```
virtual type -delete mystateType
```

- Create a virtual type that includes "don’t-cares" (the ‘-’ character).

```
virtual type {{0x01-- add}{0x02-- sub}{default bad}} mydecodetype
```

- Create a virtual type using a mask for "don’t-cares." The middle field is the mask, and the mask should have bits set to 1 for the bits that are don't care.

```
virtual type {{0x0100 0xff add}{0x0200 0xff sub}{default bad}}
mydecodetype
```

Related Topics

- virtual function
- Virtual Objects

vlib

The command creates a design library. You must use **vlib** rather than operating system commands to create a library directory or index file.

If the specified library already exists as a valid ModelSim library, the **vlib** command will exit with a warning message without touching the library.

This command provides additional information with the **-help** switch.

Syntax

vlib [-archive [-compact <percent>]] [-format { 1 | 3 }] [-dos | -short | -unix | -long] [(-lock | -unlock) <design_unit>] [-locklib | -unlocklib] <name>

Arguments

- -archive [-compact <percent>]

  (optional) Stores design units that are compiled into the created library in archives rather than in subdirectories. Refer to "Archives" for more details.

  You may optionally specify a decimal number between 0 and 1 that denotes the allowed percentage of wasted space before archives are compacted. By default archives are compacted when 50% (.5) of their space is wasted. See an example below.

  -compact — (optional) Specifies the percentage amount of wasted space before the archives are compacted where the default is 50% (0.5).

  <percent> — specified as a decimal number betwen 0 and 1.

- -format { 1 | 3 }

  (optional) Prepares a library for conversion to be compatible with a previous release, by altering the _info file.

  1 — allows you to convert a library to be compatible with the 6.2 series and earlier.

  3 — allows you to convert a library to be compatible with the 6.3 series and newer.

  The usage flow would be:

  ```
  \\1) Using a current release of the simulator, run:
          vlib -format 1 current_lib
          vcom -refresh -work current_lib
  \\    to prepare current_lib for conversion back to a 6.2 release
  \\
  \\2) Using a 6.2 release of the simulator, run:
          vcom -refresh -work current_lib
  \\    to refresh current_lib for use with the previous release
  ```

- -dos

  (optional) Specifies that subdirectories in a library have names that are compatible with DOS. Not recommended if you use the vmake utility. .

- -short

  (optional) Interchangeable with the **-dos** argument.

- -unix

  (optional) Specifies that subdirectories in a library may have long file names that are NOT compatible with DOS.

- -long

  (optional) Interchangeable with the **-unix** argument.

- (-lock | -unlock) <design_unit>

  (optional) Locks an existing design unit so it cannot be recompiled or refreshed. The -unlock switch reverses this action. File permissions are not affected by these switches.

- -locklib | -unlocklib

  (optional) Locks a complete library so that compilation cannot target the library and the library cannot be refreshed. The -unlocklib switch reverses this action. File permissions are not affected by these switches.

- <name>

  (required) Specifies the pathname or archive name of the library to be created.

Examples

- Create the design library *design*. You can define a logical name for the library using the vmap command or by adding a line to the library section of the *modelsim.ini* file that is located in the same directory.

  ```
  vlib design
  ```

- Create the design library *uut* and specifies that any design units compiled into the library are created as archives. Also specifies that each archive be compacted when 30% of its space is wasted.

  ```
  vlib -archive -compact .3 uut
  ```

vlog

The **vlog** command compiles Verilog source code and SystemVerilog extensions into a specified working library (or to the **work** library by default).

The **vlog** command may be invoked from within ModelSim or from the operating system command prompt. It may also be invoked during simulation.

Compiled libraries are major-version dependent. When moving between major versions, you have to refresh compiled libraries using the **-refresh** argument to **vlog**. This is not true for minor versions (letter releases).

All arguments to the **vlog** command are case sensitive: -WORK and -work are not equivalent.

The IEEE P1800 Draft Standard for SystemVerilog requires that the default behavior of the **vlog** command is to treat each Verilog design file listed on the command line as a separate compilation unit. This behavior is a change in **vlog** from versions prior to 6.2, wherein all files in a single command line were concatenated into a single compilation unit. To treat multiple files listed within a single command line as a single compilation unit, use either the **vlog -mfcu** argument or the MultiFileCompilationUnit *modelsim.ini* file variable.

This command provides additional information with the **-help** switch.

Syntax

vlog [options] <filename> [<filename> ...]

[options]:

[-93]

[-addpragmaprefix <prefix>]
[-compat] [-compile_uselibs[=<directory_name>]]
[-convertallparams] [-cuname]

[+define+<macro_name>[=<macro_text>]] [-deglitchalways | -nodeglitchalways]
[+delay_mode_distributed] [+delay_mode_path] [+delay_mode_unit]
[+delay_mode_zero] [-dpiforceheader] [-dpiheader <filename>]

[-E <filename>] [-Edebug <filename>] [-enumfirstinit] [-Epretty <filename>]
[-error <msg_number>[,<msg_number>,…]]

[-f <filename>] [-force_refresh <design_unit>]
[-fsmimplicittrans | -nofsmimplicittrans]
[-fsmresettrans | -nofsmresettrans] [-fsmsingle | -nofsmsingle]
[-fsmverbose[b | t | w]] [-fsmxassign | -nofsmxassign]

[-gen_xml <design_unit> <filename>]

[-hazards]

[-ignorepragmaprefix <prefix>] [+incdir+<directory>] [-incr | -noincr]
[-isymfile]

[-l <filename>] [+libcell | +nolibcell]
[+libext+<suffix>] [-libmap <pathname>] [-libmap_verbose] [+librescan]
[-line <number>] [-lint] [-lrmclassinit]

[+maxdelays] [+mindelays] [-mixedansiports]
[-mixedsvvh [b | s | v]] [-mfcu | -sfcu] [-modelsimini <ini_filepath>]

[-nodbgsym]
[-noexcludeternary <design_unit>]
[-noForceUnsignedToVhdlInteger] [-nologo] [+nospecify]
[-note <msg_number>[,<msg_number>,…]] [+notimingchecks]
[-novtblfixup] [+nowarn<CODE>] [-nowarn <category_number>]

[-oldsv] [-override_timescale <time_unit> / <time_precision>] [-O0]

[-pedanticerrors] [-permissive] [-permit_defunct_sv] [-printinfilenames]

[-quiet]

[-R [<simargs>]] [-refresh]

[-source] [-s] [-sv]
[-svext[=[+|-]<extension>[,[+|-]<extension>]*]] [-svinputport=net | var | relaxed]
[-skipprotected] [-skipprotectedmodule]
[-suppress <msg_number>[,<msg_number>,…]]
[-sv05compat] [-sv09compat] [-sv12compat]

[-time] [-timescale <time_units>/<time_precision>]
[+typdelays]

[-u]

[-v <library_file>] [-version] [-vlog01compat] [-vlog95compat] [-vmake]

[-warning <msg_number>[,<msg_number>,…]] [-work <library_name>]
[-writetoplevels <fileName>]

[-y <library_directory>]

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.

- -addpragmaprefix <prefix>

  (optional) Enables recognition of synthesis and coverage pragmas with a user specified prefix. If this argument is not specified, pragmas are treated as comments. All regular synthesis and coverage pragmas are honored.

  <prefix> — Specifies a user defined string where the default is no string, indicated by quotation marks (“”).

You may also set this with the AddPragmaPrefix variable in the vlog section of the *modelsim.ini* file.

- -compat

  Disables optimizations that result in different event ordering than Verilog-XL. Optional.

  ModelSim Verilog generally duplicates Verilog-XL event ordering, but there are cases where it is inefficient to do so. Using this option does not help you find event order dependencies, but it allows you to ignore them. Keep in mind that this option does not account for all event order discrepancies, and that using this option may degrade performance. Refer to "Event Ordering in Verilog Designs" for additional information.

- -compile_uselibs[=<directory_name>]

  Locates source files specified in a **`uselib** directive (Refer to "Verilog-XL uselib Compiler Directive"), compiles those files into automatically created libraries, and updates the *modelsim.ini* file with the logical mappings to the new libraries. Optional. If a *directory_name* is not specified, ModelSim uses the name specified in the MTI_USELIB_DIR environment variable. If that variable is not set, ModelSim creates the directory *mti_uselibs* in the current working directory.

- -convertallparams

  Enables converting parameters not defined in ANSI style to VHDL generics of type std_logic_vector, bit_vector, std_logic, vl_logic, vl_logic_vector, and bit. Optional.

- -cuname

  Used only in conjunction with **-mfcu**. Optional. The **-cuname** names the compilation unit being created by **vlog**. The named compilation unit can then be specified on the vsim command line, along with the <top> design unit. The purpose of doing so is to force elaboration of specified compilation unit package, thereby forcing elaboration of a necessary 'bind' statement within that compilation unit that would otherwise not be elaborated. An example of the necessary commands is:

  ```
  vlog -cuname pkg_name -mfcu file1.sv file2.sv
  vsim top pkg_name
  ```

  You need to do this only in cases where you have a 'bind' statement in a module that might otherwise not be elaborated, because no module in the design depends on that compilation unit. In other words, if a module that depends on that compilation unit exists, you don't need to force the elaboration, for it occurs automatically. Also, if you are using qverilog to compile and simulate the design, this binding issue is handled properly automatically.

- +define+<macro_name>[=<macro_text>]

  Allows you to define a macro from the command line that is equivalent to the following compiler directive:

  ```
  `define <macro_name> <macro_text>
  ```

  Optional. You can specify more than one macro with a single +**define**. For example:

```
vlog +define+one=r1+two=r2+three=r3 test.v
```

A command line macro overrides a macro of the same name defined with the **`define** compiler directive.

- -deglitchalways | -nodeglitchalways

  Reduces the incidents of zero delay oscillations among always_comb and always @* combinatorial logic blocks that produce glitches on the variables they write. This is the default behavior. Old behavior may be selected with the vlog -nodeglitchalways option. A side effect of this behavior is that time zero races involving the glitch-producing always blocks may resolve in a different order.

- +delay_mode_distributed

  Disables path delays in favor of distributed delays. Optional. Refer to "Delay Modes" for details.

- +delay_mode_path

  Sets distributed delays to zero in favor of using path delays. Optional.

- +delay_mode_unit

  Sets path delays to zero and non-zero distributed delays to one time unit. Optional.

- +delay_mode_zero

  Sets path delays and distributed delays to zero. Optional.

- -dpiforceheader

  (optional) Forces the generation of a DPI header file even if it will be empty of function prototypes.

- -dpiheader <filename>

  (optional) Generates a header file that may then be included in C source code for DPI import functions. Refer to "DPI Use Flow" for additional information.

- -E <filename>

  (optional) Captures text processed by the Verilog parser after preprocessing has occurred and copies that text to an output file. This includes text read from source files specified by using the -v or -y argument.

  <filename> — Specifies a name for the debugging output file. Wildcards are not allowed.

  Generally, preprocessing consists of the following compiler directives: `ifdef, `else, `elsif, `endif, `ifndef, `define, `undef, `include.

  The `line directive attempts to preserve line numbers, file names, and level in the output file (per the 1800-2009 LRM). White space is usually preserved, but sometimes it may be deleted or added to the output file.

- -Edebug <filename>

  (optional) Captures text processed by the Verilog parser after preprocessing has occurred and copies that text to a debugging output file.

  <filename> — Specifies a name for the debugging output file. Wildcards are not allowed.

  Generally, preprocessing consists of the following compiler directives: `ifdef, `else, `elsif, `endif, `ifndef, `define, `undef, `include. The file is a concatenation of source files with `include expanded. The file can be compiled and then used to find errors in the original source files. The `line directive attempts to preserve line numbers and file names in the output file. White space is usually preserved, but sometimes it may be deleted or added to the output file.

- -enumfirstinit

  (optional) Initializes enum variables in SystemVerilog using the leftmost value as the default. You must also use the argument with the vsim command in order to implement this initialization behavior. Specify the EnumBaseInit variable as 0 in the *modelsim.ini* file to set this as a permanent default.

- -Epretty <filename>

  Captures text processed by the Verilog parser after preprocessing has occurred, performs some formatting for better readability, and copies that text to an output file, <filename>. Optional.

- -error <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "error." Optional. Edit the error variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- -f <filename>

  Specifies a file with more command line arguments. Optional. Allows complex arguments to be reused without retyping. Allows gzipped input files. Nesting of **-f** options is allowed.

  Refer to the section "Argument Files" for more information.

- -force_refresh <design_unit>

  Forces the refresh of all specified design units. Optional. By default, the work library is updated; use **-work <library_name>**, in conjunction with **-force_refresh**, to update a different library (for example, vlog -work <your_lib_name> -force_refresh).

  When the compiler refreshes a design unit, it checks each dependency to ensure its source has not been changed and recompiled. Sometimes the dependency checking algorithm changes from release to release. This can lead to false errors during the integrity checks performed by the **-refresh** argument. An example of such a message follows:

  ```
  ** Error: (vsim-13) Recompile /u/test/dware/dware_61e_beta.dwpackages
  because /home/users/questasim/linux/../synopsys.attributes has changed.
  ```

The **-force_refresh** argument forces the refresh of the design unit, overriding any dependency checking errors encountered by the **-refresh** argument.

A more conservative approach to working around **-refresh** dependency checks is to recompile the source code, if it is available.

- -fsmimplicittrans | -nofsmimplicittrans

  (optional) Toggles recognition of implied same state transitions. This setting is off by default.

- -fsmresettrans | -nofsmresettrans

  (optional) Toggles recognition of synchronouos or asynchronous reset transitions. This includes/excludes reset transitions in coverage results. This setting is on by default.

- -fsmsingle | -nofsmsingle

  (optional) Toggles the recognition of VHDL FSMs where the current state variable of type std_logic, bit, boolean, or single-bit std_logic_vector/bit_vector and Verilog single-bit FSMs. This setting is off by default.

- -fsmverbose[b | t | w]

  Provides information about FSMs detected, including state reachability analysis. Optional.

  b — displays only basic information.

  t — displays a transition table in addition to the basic information.

  w — displays any warning messages in addition to the basic information.

  When you do not specify an argument, this switch reports all information similar to:

```
# ** Note: (vlog-1947)   FSM RECOGNITION INFO
#      Fsm detected in : ../fpu/rtl/vhdl/serial_mul.vhd
#      Current State Variable : s_state :
../fpu/rtl/vhdl/serial_mul.vhd(76)
#      Clock : clk_i
#      Reset States are: { waiting , busy }
#      State Set is : { busy , waiting }
#      Transition table is
#         ----------------------------------------
#         busy      =>   waiting Line : (114 => 114)
#         busy      =>   busy    Line : (111 => 111)
#         waiting   =>   waiting Line : (120 => 120) (114 => 114)
#         waiting   =>   busy    Line : (111 => 111)
#         ----------------------------------------
```

  When you do not specify this switch, you will receive a message similar to:

```
# ** Note: (vlog-143) Detected '1' FSM/s in design unit
'serial_mul.rtl'.
```

- -fsmxassign | -nofsmxassign

  (optional) Toggles recognition of finite state machines (FSMs) containing X assignment. This option is used to detect FSMs if current state variable or next state variable has been assigned "X" value in a "case" statement. FSMs containing X-assign are otherwise not detectable. This setting is on by default.

- -gen_xml <design_unit> <filename>

  Produces an XML-tagged file containing the interface definition of the specified module. Optional. This option requires a two-step process where you must 1) compile <filename> into a library with **vlog** (without **-gen_xml**) then 2) execute **vlog** with the **-gen_xml** switch, for example:

  ```
  vlib work
  vlog counter.v
  vlog -gen_xml counter counter.xml
  ```

- -hazards

  Detects event order hazards involving simultaneous reading and writing of the same register in concurrently executing processes. Optional. You must also specify this argument when you simulate the design with vsim. Refer to "Hazard Detection" for more details.

Note

Enabling **-hazards** implicitly enables the **-compat** argument. As a result, using this argument may affect your simulation results.

- -ignorepragmaprefix <prefix>

  (optional) Directs vlog to ignore synthesis and coverage pragmas with the specified prefixname. All affected pragmas will be treated as regular comments. Edit the IgnorePragmaPrefix *modelsim.ini* variable to set a permanent default.

  <prefix> — Specifies a user defined string.

- +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 compilation. 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 of whether you use vlog -incr or not.

- -isymfile

  Generates a complete list of all imported tasks and functions (TFs). Used with DPI to determine all imported TFs that are expected by ModelSim.

- -l <filename>

  (optional) Generates a log file of the compile.

- +libcell | +nolibcell

  +libcell — Treats all modules found and compiled by source library search as though they contained a ‘celldefine compiler directive, thus marking them as cells (refer to the -v and -y arguments of vlog, which enable source library search). Using the +libcell argument matches historical behavior of Verilog-XL with respect to source library search. Optional.

  +nolibcell — (default) Disables treating all modules found and compiled by source library search as though they contained a ‘celldefine compiler directive. That is, this argument restores the default library search behavior if you have changed it using the +libcell | +nolibcell argument. Optional.

Note

By default, wildcard logging and code coverage exclude cells. For more information, refer to the -nocovercells and -covercells arguments of vlog and to the description of wildcard logging performed by the **log** command.

- +libext+<suffix>

  Works in conjunction with the **-y** option. Specifies file extensions for the files in a source library directory. Optional. By default, the compiler searches for files without extensions. If you specify the +libext argument, then the compiler will search for a file with the suffix appended to an unresolved name. You may specify only one +libext option, but it may contain multiple suffixes separated by the plus character (+). The extensions are tried in the order you specify them with the +libext argument.

- -libmap <pathname>

  Specifies a Verilog 2001 library map file. Optional. You can omit this argument by placing the library map file as the first option on the vlog invocation (e.g., *vlog top.map top.v top_cfg.v*).

- -libmap_verbose

  Displays library map pattern matching information during compilation. Optional. Use to troubleshoot problems with matching filename patterns in a library file.

- +librescan

  Scans libraries in command-line order for all unresolved modules. Optional.

- -line <number>

  Starts the compiler on the specified line in the Verilog source file. Optional. By default, the compiler starts at the beginning of the file.

- -lint

  (optional) Issues warnings on the following lint-style static checks:

  - when Module ports are NULL.
  - when assigning to an input port.

- when referencing undeclared variables/nets in an instantiation.

This switch generates additional array bounds-checking code, which can slow down simulation, to check for the following:

- index warnings for dynamic arrays
- when an index for a Verilog unpacked variable array reference is out of bounds.

The warnings are reported as WARNING[8]. You can also enable this option using the Show_Lint variable in the *modelsim.ini* file.

- -lrmclassinit

  Changes initialization behavior to match the SystemVerilog specification (per IEEE Std 1800-2007) where all superclass properties will be initialized before any subclass properties.

- +maxdelays

  Selects maximum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

- +mindelays

  Selects minimum delays from the "min:typ:max" expressions. Optional. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

- -mixedansiports

  Permits partial port redeclarations.

- -mixedsvvh [b | s | v]

  Facilitates using SystemVerilog packages at the SystemVerilog-VHDL boundary of a mixed-language design. When you compile a SystemVerilog package with -mixedsvvh, the package can be included in a VHDL design as if it were defined in VHDL itself. Optional.

  b — treats all scalars/vectors in the package as VHDL bit/bit_vector

  s — treats all scalars/vectors in the package as VHDL std_logic/std_logic_vector

  v — treats all scalars/vectors in the package as VHDL vl_logic/vl_logic_vector

- -mfcu

  Instructs the compiler to treat all files within a compilation command line as a single compilation unit. Optional. The default behavior is to treat each file listed in a command as a separate compilation unit, per the SystemVerilog standard. Prior versions concatenated the contents of the multiple files into a single compilation unit by default. You can also enable this option using the MultiFileCompilationUnit variable in the *modelsim.ini* file.

- -modelsimini <ini_filepath>

  Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

- -nodbgsym

  Disables the generation of the symbols debugging database in the compiled library.

  The symbols debugging database is the .dbs file in the compiled library that provides information to the GUI allowing you to view detailed information about design objects at the source level. Two major GUI features that use this database include source window annotation and textual dataflow.

  You should only specify this switch if you know that anyone using the library will not require this information for design analysis purposes.

- -noexcludeternary <design_unit>

  (optional) Disables the automatic exclusion of UDB coverage data rows resulting from ternary expressions for the specified design unit. Normal operation for code coverage is to include rows corresponding to the case where two data inputs are the same, and the select input is a “don’t care”. To disable this automatic exclusion for the entire design, use “vsim -noexcludeternary” instead.

- -noForceUnsignedToVhdlInteger

  Prevents untyped Verilog parameters in mixed-language designs that are initialized with unsigned values between 2*31-1 and 2*32 from being converted to a VHDL generic. By default, untyped Verilog parameters that are initialized with unsigned values between 2*31-1 and 2*32 are converted to VHDL INTEGER generics. Because VHDL INTEGER parameters are signed numbers, the Verilog values 2*31-1 to 2*32 are converted to negative VHDL values in the range from -2*31 to -1 (the 2's complement value).

- -noincr

  Disables incremental compilation previously turned on with -incr argument. Optional. Default.

- -nologo

  Disables the startup banner. Optional.

- +nospecify

  Disables specify path delays and timing checks. Optional.

- -note <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "note." Optional. Edit the note variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- +notimingchecks

  Removes all timing check entries from the design as it is parsed. Optional.

- -novtblfixup

  Causes virtual method calls in SystemVerilog class constructors to behave as they would in normal class methods, which prevents the type of a `this` reference from changing during construction.

  This overrides default behavior, where the type of a `this` reference is treated as if it is a handle to the type of the active `new()` method while a constructor is executing (which implies that virtual method calls resolve will not execute methods of an uninitialized class type).

- +nowarn<CODE>

  Disables warning messages in the category specified by <CODE>. Optional. Warnings that can be disabled include the <CODE> name in square brackets in the warning message. For example,

  ```
  ** Warning: test.v(15): [RDGN] - Redundant digits in numeric
  literal.
  ```

  This warning message can be disabled by specifying +nowarnRDGN.

- -nowarn <category_number>

  Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. Multiple **-nowarn** switches are allowed. Warnings may be disabled for all compiles via the Main window **Compile > Compile Options** menu command or the *modelsim.ini* file (refer to modelsim.ini Variables).

  The warning message categories are described in Table 2-7:

Table 2-7. Warning Message Categories for vlog -nowarn

Category number	Description
12	non-LRM compliance in order to match Cadence behavior
13	constructs that code coverage can't handle

- -oldsv

  (optional) Allows unpacked array concatenation on input ports to be treated as an assignment pattern. Accepts constant expressions in variable loop list with the standard foreach syntax – foreach (array[i, j, k]) – and alternate foreach syntax – foreach (array[i][j][k]).

- -override_timescale <time_unit> / <time_precision>

  Specifies a timescale for all compiled design units. This timescale overrides all ‘timescale directives and all declarations of timeunit and timeprecision. Optional.

time_unit — unit of measurement for times and delays. This specification consists of one of three integers (1, 10, or 100) representing order of magnitude and one of six character strings representing units of measurement:

{1 | 10 | 100} {s | ms | us | ns | ps | fs}

For example, 10 ns.

time_precision — unit of measurement for rounding delay values before being used in simulation. Allowable values are the same as for time_unit.

- -O0

  Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

- -pedanticerrors

  (optional) Enforces strict compliance of the IEEE Std 1800-2005 in the following cases:

  - Covergroup bin size, value range, or transition specification must be constant.
  - Using "new” for queues is not legal. When strict compliance is not enforced, use of "new" creates a queue of the specified size where all elements are initialized to the default value of the queue element type.
  - Using underscore character (_) in sized, based literals is not legal. When you specify this argument, an error will occur for literals such as 2'b_01.
  - Omitting the grave accent mark (`) preceding the left brace ({) when writing structure literals is not legal. When you specify this argument, an error will occur for literals written without that mark.
  - Inserting the grave accent mark to precede quotation marks (`") that enclose string literals is not legal—only string literals within quotation marks (") are allowed. When you specify this argument, an error will occur for string literals using that mark.
  - Using class extern method prototypes with lifetime (automatic/static) designations produces a compliance error (instead of a warning).
  - Using “cover bool@clk” as a PSL statement.
  - Using realtime data types in SystemVerilog assertions.
  - Using an unsized constant in a concatenation if it is the leftmost value in the list.

  This argument also produces a report of mismatched ‘else directives.

- -permissive

  Allows messages in the LRM group of error messages to be downgraded to a warning. Optional.

- -permit_defunct_sv

  Allows using a selected set of constructs no longer supported by the SystemVerilog standard. Currently, the set supports only the use of the keyword “char.” This argument allows use of the keyword “char” to be interpreted as the SystemVerilog “byte” type. Optional.

- -printinfilenames

  Prints the path names of all source files opened (including “include” files) during the compile. Specifies whether each file is a Verilog or SystemVerilog file.

- -quiet

  Disables 'Loading' messages. Optional.

- -R [<simargs>]

  Instructs the compiler to invoke vsim after compiling the design. The compiler automatically determines which top-level modules are to be simulated. The command line arguments following -R are passed to the simulator, not the compiler. Place the -R option at the end of the command line or terminate the simulator command line arguments with a single "-" character to differentiate them from compiler command line arguments.

  The -R option is not a Verilog-XL option, but it is used by ModelSim to combine the compile and simulate phases together as you may be used to doing with Verilog-XL. It is not recommended that you regularly use this option because you will incur the unnecessary overhead of compiling your design for each simulation run. Mainly, it is provided to ease the transition to ModelSim.

- -refresh

  Regenerates a library image. Optional. By default, the work library is updated. To update a different library, use **-work <library_name>** with **-refresh** (for example, vlog -work <your_lib_name> -refresh). If a dependency checking error occurs which prevents the refresh, use the **vlog -force_refresh** argument. See **vlog** examples for more information. You may use a specific design name with **-refresh** to regenerate a library image for that design, but you may not use a file name.

- -sfcu

  Instructs the compiler to treat all files within a compilation command line as a separate compilation units. This is the default behavior and is the inverse of the behavior of -mfcu.

  This switch will override the MultiFileCompilationUnit variable if it is set to "1" in the *modelsim.ini* file.

- -source

  Displays the associated line of source code before each error message that is generated during compilation. Optional; by default, only the error message is displayed.

- -s

  Instructs the compiler not to load the **standard** package. Optional. This argument should only be used when you are compiling the **sv_std** package.

- -sv

  Enables SystemVerilog features and keywords. Optional. By default ModelSim follows the IEEE Std 1364-2001 and ignores SystemVerilog keywords. If a source file has a ".sv" extension, ModelSim will automatically parse SystemVerilog keywords.

- -svext[=[+|-]<extension>[,[+|-]<extension>]*]

  (optional) Enables SystemVerilog language extensions.

  Where <extension> is one of the following:

  feci — Treat constant expressions in a foreach loop variable index as constant.

  pae — Automatically export all symbols imported and referenced in a package.

  uslt — (default) Promote unused design units found in source library files specified with the -y option to top-level design units.

  spsl — (default) Search for packages in source libraries specified with -y and +libext.

  Multiple extensions are specified as a comma separated list. For example:

  vlog -svext=-feci,+uslt,spsl

- -svinputport=net | var | relaxed

  Used in conjunction with -sv to determine the default data type assigned to an input port declaration. Optional.

  net — declares the port to be a net. This value enforces strict compliance to the Verilog LRM (IEEE Std 1364-2005), where the port declaration defaults to wire.

  var — declares the port to be a variable. This value enforces behavior from previous releases, where the port declaration defaults to variable.

  relaxed — (default) declares the port to be a net only if the type is a 4-state scalar or 4-state single dimensional vector. Otherwise, the port is declared a variable.

- -sv05compat

  Used in conjunction with the -sv switch to ensure compatibility with the reserved keyword set of IEEE Std 1800-2005.

- -sv09compat

  Used in conjunction with the -sv switch to ensure compatibility with the reserved keyword set of IEEE Std 1800-2009.

- -sv12compat

  Used in conjunction with the -sv switch to ensure compatibility with the reserved keyword set of IEEE Std 1800-2012.

- -skipprotected

  Ignores any ‘protected/‘endprotected region contained in a module. Optional.

- -skipprotectedmodule

  Prevents adding any module containing a ‘protected/‘endprotected region to the library. Optional.

- -suppress <msg_number>[,<msg_number>,…]

  Prevents the specified message(s) from displaying. The <msg_number> is the number preceding the message you wish to suppress. Optional. You cannot suppress Fatal or Internal messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing message Severity Level” for more information.

- -time

  Reports the "wall clock time" **vlog** takes to compile the design. Optional. Note that if many processes are running on the same system, wall clock time may differ greatly from the actual "cpu time" spent on **vlog**.

- -timescale <time_units>/<time_precision>

  Specifies the default timescale for modules not having an explicit timescale directive in effect during compilation. Optional. The format of the -timescale argument is the same as that of the `timescale directive. The format for <*time_units*> and <*time_precision*> is <*n*><*units*>. The value of <*n*> must be 1, 10, or 100. The value of <*units*> must be fs, ps, ns, us, ms, or s. In addition, the <*time_units*> must be greater than or equal to the <*time_precision*>.

- +typdelays

  Selects typical delays from the "min:typ:max" expressions. Default. If preferred, you can defer delay selection until simulation time by specifying the same option to the simulator.

- -u

  Converts regular Verilog identifiers to uppercase. Allows case insensitivity for module names. Optional.

- -v <library_file>

  Specifies a source library file containing module and UDP definitions. Optional. Refer to “Verilog-XL Compatible Compiler Arguments” for more information.

  After all explicit filenames on the **vlog** command line have been processed, the compiler uses the **-v** option to find and compile any modules that were referenced but not yet defined. Modules and UDPs within the file are compiled only if they match previously unresolved references. Multiple **-v** options are allowed. See additional discussion in the examples.

- -version

  Returns the version of the compiler as used by the licensing tools. Optional.

- -vlog01compat

  Ensures compatibility with rules of IEEE Std 1364-2001. Default.

- -vlog95compat

  Disables Verilog 2001 keywords, which ensures that code that was valid according to the 1364-1995 spec can still be compiled. By default ModelSim follows the rules of IEEE Std 1364-2001. Some requirements in 1364-2001 conflict with requirements in 1364-1995. Optional. Edit the vlog95compat variable in the *modelsim.ini* file to set a permanent default.

- -vmake

  Generates a complete record of all command line data and files accessed during the compile of a design. This data is then used by the vmake command to generate a comprehensive makefile for recompiling the design library. By default, vcom stores compile data needed for the -refresh switch and ignores compile data not needed for -refresh. The -vmake switch forces inclusion of all file dependencies and command line data accessed during a compile, whether they contribute data to the initial compile or not. Executing this switch can increase compile time in addition to increasing the accuracy of the compile. See the vmake command for more information.

- -warning <msg_number>[,<msg_number>,…]

  Changes the severity level of the specified message(s) to "warning." Optional. Edit the warning variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -work <library_name>

  Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional; by default, the compiled design units are added to the **work** library. The specified pathname overrides the pathname specified for work in the project file.

- -writetoplevels <fileName>

  (optional) Records the names of all top level module names in a specified file. Also records any compilation unit name specified with **-cuname**. May only be specified when compiling the top level modules.

  <fileName> — Required. Specifies the name of the file where module names are to be recorded.

- -y <library_directory>

  Specifies a source library directory containing definitions for modules, packages, interfaces, and user-defined primitives (UDPs). Usually, this is a directory of source files that you want to scan if the compiled versions do not already exist in a library. Optional. Refer to "Verilog-XL Compatible Compiler Arguments" for more information.

  After all explicit filenames on the **vlog** command line have been processed, the compiler uses the **-y** option to find and compile any modules that were referenced but not yet defined. Files within this directory are compiled only if the file names match the names of previously unresolved references. Multiple **-y** options are allowed. You will need to specify a file suffix

by using **-y** in conjunction with the **+libext+<suffix>** option. See additional discussion in the examples.

Note

Any **-y** arguments that follow a **-refresh** argument on a **vlog** command line are ignored. Any **-y** arguments that come before the **-refresh** argument on a **vlog** command line are processed.

- <filename>

  Specifies the name of the Verilog source code file to compile. One filename is required. Multiple filenames can be entered separated by spaces. Wildcards can be used.

Examples

- Compile the Verilog source code contained in the file *example.vlg*.

  ```
  vlog example.vlg
  ```

- After compiling *top.v*, **vlog** will scan the file *und1* for modules or primitives referenced but undefined in *top.v*. Only referenced definitions will be compiled.

  ```
  vlog top.v -v und1
  ```

- After compiling *top.v*, **vlog** will scan the *vlog_lib* library for files with modules with the same name as primitives referenced, but undefined in *top.v*. The use of **+libext+.v+.u** implies filenames with a *.v* or *.u* suffix (any combination of suffixes may be used). Only referenced definitions will be compiled.

  ```
  vlog top.v +libext+.v+.u -y vlog_lib
  ```

  The **-work** option specifies **mylib** as the library to regenerate. **-refresh** rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of ModelSim.

- If your library contains VHDL design units, be sure to regenerate the library with the vcom command using the **-refresh** option as well. Refer to "Regenerating Your Design Libraries" for more information.

  ```
  vlog -work mylib -refresh
  ```

- The **-incr** option determines whether or not the module source or compile options have changed as *module1.v* is parsed. If no change is found, the code generation phase is skipped. Differences in compile options are determined by comparing the compiler options stored in the _info file with the compiler options given. They must match exactly.

  ```
  vlog module1.v -u -O0 -incr
  ```

- The **-timescale** option specifies the default timescale for *module1.v*, which did not have an explicit timescale directive in effect during compilation. Quotes (" ") are necessary because the argument contains white spaces.

```
vlog module1.v -timescale "1 ns / 1 ps"
```

vmake

The **vmake** utility allows you to use a UNIX or Windows MAKE program to maintain individual libraries. You run **vmake** on a compiled design library. This utility operates on multiple source files per design unit; it supports Verilog include files as well as Verilog and VHDL PSL vunit files.

Note

If a design is spread across multiple libraries, then each library must have its own makefile and you must build each one separately.

By default, the output of **vmake** is sent to stdout—however, you can send the output to a makefile by using the shell redirect operator (>) along with the name of the file. You can then run the makefile with a version of MAKE (not supplied with ModelSim) to reconstruct the library. *This command must be invoked from either the UNIX or the Windows/DOS prompt.*

A MAKE program is included with Microsoft Visual C/C++, as well as many other program development environments.

After running the **vmake** utility, MAKE recompiles only the design units (and their dependencies) that have changed. You run **vmake** only once; then you can simply run MAKE to rebuild your design. If you add new design units or delete old ones, you should re-run **vmake** to generate a new makefile.

The **vmake** utility ignores library objects compiled with **-nodebug**.

This command provides additional information with the **-help** switch.

Syntax

vmake [-cygdrive] [-du <design_unit_name> ...] [-f <filename>] [-fullsrcpath] [-ignore] [<library_name>] [-modelsimini <ini_filepath>]

Arguments

- -cygdrive

  Generates a makefile that uses a path specified with UNIX pathname conventions. Use this argument if you are using cygwin v3.81 or later (which no longer supports Windows conventions for drive and pathname). Optional.

- -du <design_unit_name>

  Specifies that a vmake file will be generated only for the specified design unit. You can specify this argument any number of times for a single vmake command. Optional.

- -f <filename>

  Specifies a file to read command line arguments from. Optional.

  Refer to the section "Argument Files" for more information

- -fullsrcpath

  Produces complete source file paths within generated makefiles. By default, source file paths are relative to the directory in which compilations originally occurred. Use this argument to copy and evaluate generated makefiles within directories that are different from where compilations originally occurred. Optional.

- -ignore

  Omits a make rule for the named primary design unit and its secondary design units. Optional.

- <library_name>

  Specifies the library name; if none is specified, then work is assumed. Optional.

- -modelsimini <ini_filepath>

  Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified by the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems, the path separator should be a forward slash (/). Optional.

Examples

- To produce a makefile for the work library:

  ```
  vmake >mylib.mak
  ```

- To run **vmake** on libraries other than **work**:

  ```
  vmake mylib >mylib.mak
  ```

- To rebuild **mylib,** specify its makefile when you run MAKE:

  ```
  make -f mylib.mak
  ```

- To use **vmake** and MAKE on your **work** library:

  ```
  C:\MIXEDHDL> vmake >makefile
  ```

- To edit an HDL source file within the work library:

  ```
  C:\MIXEDHDL> make
  ```

  Your design gets recompiled for you. You can change the design again and re-run MAKE to recompile additional changes.

- To run **vmake** on libraries other than **work**:

  ```
  C:\MIXEDHDL> vmake mylib >mylib.mak
  ```

- To rebuild **mylib,** specify its makefile when you run MAKE:

  ```
  C:\MIXEDHDL> make -f mylib.mak
  ```

vmap

The **vmap** command defines a mapping between a logical library name and a directory by modifying the *modelsim.ini* file.

With no arguments, **vmap** reads the appropriate *modelsim.ini* file(s) and prints to the transcript the current logical library to physical directory mappings.

This command provides additional information with the **-help** switch.

Syntax

vmap [-c | -del <logical_name> ... | <logical_name> [<path>]]
[-modelsimini <path/modelsim.ini>]

Arguments

- -c

  (optional) Copies the default *modelsim.ini* file from the ModelSim installation directory to the current directory.

  This argument is intended only for making a copy of the default *modelsim.ini* file to the current directory. Do not use it while making your library mappings or the mappings may end up in the incorrect copy of the *modelsim.ini*.

- -del <logical_name> ...

  (optional) Deletes the mapping specified by <logical_name> from the current project file. You can specify multiple logical name arguments to the -del switch to delete multiple library mappings.

- <logical_name> [<path>]

  (optional) Maps a logical library name to the specified physical library.

  If you do not specify <path> the command returns the current mapping for <logical_name>.

- -modelsimini <path/modelsim.ini>

  (optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

Examples

- Map two logical libraries to the physical library “work”, then delete the two logical libraries:

  ```
  vlib work
  vmap library1 work
  vmap library2 work
  ```

- Display information about the logical library “library1”:

```
vmap library1
```

- Delete the logical library mappings:

```
vmap -del library1 library2
```

vsim

The **vsim** command invokes the VSIM simulator, which you can use to view the results of a previous simulation run (when invoked with the **-view** switch), or to view coverage data stored in the UCDB from a previous simulation run (when invoked with the **-viewcov** switch).

You can simulate a VHDL configuration or an entity/architecture pair, a Verilog module or configuration. 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.

You can use this command in batch mode from the Windows command prompt. Refer to "Batch Mode" for more information on the VSIM batch mode.

To manually interrupt design loading, use the Break key or <Ctrl-C> from a shell.

You can invoke **vsim** from a command prompt or in the Transcript window of the Main window. You can also invoke it from the GUI by selecting Simulate > Start Simulation.

All arguments to the **vsim** command are case sensitive; for example, -g and -G are not equivalent.

Syntax

Note

This Syntax section presents all of the vsim switches in alphabetical order, while the Arguments section groups the arguments into the following sections:

Arguments, all languages
Arguments, VHDL
Arguments, Verilog
Arguments, object

vsim [options]

[options]:

[-absentisempty] [+alt_path_delays] [-assertfile <filename>]

[+bitblast[=[iopath | tcheck]]]

[-c] [-capacity][-classdebug] [-colormap new]

[-debugdb=<db_pathname>] [-defaultstdlogicinittoz] [+delayed_timing_checks]
[-display <display_spec>] [-displaymsgmode both | tran | wlf]
[-do "<command_string>" | <macro_file_name>] [-donotcollapsepartiallydriven]
[-dpiforceheader]
[-dpiforceheader] [-dpiheader] [-dpilib <libname>] [-dpioutoftheblue 0 | 1 | 2]
[+dumpports+collapse | +dumpports+nocollapse] [+dumpports+direction]
[+dumpports+no_strength_range] [+dumpports+unique]

[-error <msg_number>[,<msg_number>,…]]
[-enumfirstinit]
[-errorfile <filename>]

[-f <filename>] [-fatal <msg_number>[,<msg_number>,…]]

[-g <Name>=<Value> …] [-G<Name>=<Value> …] [-gblso <filename>]
[-geometry <geometry_spec>] [-gui]

[-hazards] [-help]

[-i]
[+initregNBA] [-installcolormap]

[-keeploaded] [-keeploadedrestart] [-keepstdout]

[-l <filename>] [-L <library_name> …] [-lib <libname>]
[<library_name>.<design_unit>] [-Lf <library_name> …]

[+maxdelays] [+mindelays]
[-modelsimini <ini_filepath>]
[-msglimit <msg_number>[,<msg_number>,…]] [-msgmode both | tran | wlf]
[-multisource_delay min | max | latest] [+multisource_int_delays]

[-name <name>] [+no_autodtc] [-noautoldlibpath] [-nodpiexports]
[+no_cancelled_e_msg] [+no_glitch_msg] [+no_neg_tchk] [+no_notifier]
[+no_path_edge] [+no_pulse_msg] [-no_risefall_delaynets]
[+no_show_cancelled_e] [+no_tchk_msg] [-nocollapse] [-nocapacity] [-nocompress]
[-noexcludehiz] [-noexcludeternary] [-nofileshare]
[-noimmedca] [-noglitch] [-noschematic]
[+nosdferror] [+nosdfwarn] [+nospecify] [-nostdout]
[-note <msg_number>[,<msg_number>,…]]
[+notimingchecks | +ntcnotchks] [-novhdlvariablelogging] [+nowarnBSOB]
[+nowarn<CODE | number>] [-nowiremodelforce] [+ntc_warn] [+ntcnotchks]

[-oldvhdlforgennames] [-onfinish ask | stop | exit | final]

[-pduignore[=<instpath>]] [-permissive] [-pli "<object list>"]
[-plicompatdefault [latest | 2005 | 2001]] [+<plusarg>] [-printsimstats]
[+pulse_e/<percent>] [+pulse_e_style_ondetect] [+pulse_e_style_onevent]
[+pulse_r/<percent>]

[-quiet]

[-runinit]

[+sdf_iopath_to_prim_ok]
[+sdf_nocheck_celltype]
[-sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=]<sdf_filename>]
[-sdfminr | -sdftypr | -sdfmaxr[@<delayScale>] [<instance>=]<sdf_filename>]
[-sdfmaxerrors <n>] [-sdfnoerror] [-sdfnowarn] [+sdf_report_unannotated_insts]
[+sdf_verbose] [-std_input <filename>] [-std_output <filename>]
[+show_cancelled_e]
[-strictvital] [-suppress <msg_number>[,<msg_number>,…]] [-sv_lib <shared_obj>]
[-sv_liblist <filename>] [-sv_root <dirname>]
[-sync]

[-t [<multiplier>]<time_unit>] [-tab <tabfile>] [-tag <string>] [-title <title>]
[-trace_foreign <int>] [+transport_int_delays]
[+transport_path_delays] [+typdelays]

[-usenonstdcoveragesavesysf]
[-uvmcontrol={<args>}]

[-v2k_int_delays][-vcdstim [<instance>=]<filename>]
[-version] [-vhdlvariablelogging] [-view [<alias_name>=]<WLF_filename>]
[-viewcov [<dataset_name>=]<UCDB_filename>] [-visual <visual>] [-vital2.2b]

[-warning <msg_number>[,<msg_number>,…]] [-wlf <file_name>]
[-wlfcachesize <n>] [-wlfcollapsedelta] [-wlfcollapsetime] [-nowlfcollapse]
[-wlfcompress] [-nowlfcompress] [-wlfdeleteonquit] [-nowlfdeleteonquit]
[-wlflock] [-nowlflock] [-wlfopt] [-nowlfopt] [-wlfsimcachesize <n>] [-wlfslim <size>]
[-wlftlim <duration>]

Arguments, all languages

- -assertfile <filename>

  (optional) Designates an alternative file for recording VHDL assertion messages. An alternate file may also be specified by the AssertFile *modelsim.ini* variable. By default, assertion messages are output to the file specified by the TranscriptFile variable in the *modelsim.ini* file. Refer to "Creating a Transcript File" for more information.

- +bitblast[=[iopath | tcheck]]

  (optional) Enables bit-blasting of specify block iopaths and timing checks (tchecks) with wide atomic ports. Without the optional qualifiers, the switch operates on both specify paths and tchecks. The qualifiers work as follows:

  +bitblast=iopath — bit-blasts only specify paths with wide ports.

  +bitblast=tcheck — bit-blasts only tchecks with wide ports.

  This switch is intended for use with applications employing SDF annotation.

- -c

  (optional) Specifies that the simulator is to be run in command-line mode. Refer to "Modes of Operation" for more information.

- -capacity

  (optional) Enables the fine-grain analysis display of memory capacity where the default is coarse-grain analysis display.

- -colormap new

  (optional) Specifies that the window should have a new private colormap instead of using the default colormap for the screen.

- -debugdb=<db_pathname>

  (optional) Instructs ModelSim to generate a database of connectivity information to be used for post-sim debug in the Dataflow and Schematic windows. The database pathname should have a *.dbg* extension. If a database pathname is not specified, ModelSim creates a database file named *vsim.dbg* in the current directory.

An existing *.dbg* file will be reused and a note printed to the transcript when the -debugdb switch is specified and your design has not changed since the database was created.

Refer to "Post-Simulation Debug Flow Details" for more information.

- -defaultstdlogicinittoz

  (optional) Sets the default VHDL initialization of std_logic to "Z" (high impedance) for ports of type OUT and INOUT. IEEE Std 1076-1987 VHDL Language Reference Manual (LRM) compliant behavior is for std_logic to initialize to "U" (uninitialized) which is incompatible with the behavior expected by synthesis and hardware.

- -display <display_spec>

  (optional) Specifies the name of the display to use. Does not apply to Windows platforms.

  For example:

  ```
  -display :0
  ```

- -displaymsgmode both | tran | wlf

  (optional) Controls the transcription of $display system task messages to the transcript and/or the Message Viewer. Refer to the section "Message Viewer Window" in the User's Manual for more information and the displaymsgmode *.ini* file variable.

  both — outputs messages to both the transcript and the WLF file.

  tran — outputs messages only to the transcript, therefore they are not available in the Message Viewer. Default behavior.

  wlf — outputs messages only to the WLF file/Message Viewer, therefore they are not available in the transcript.

  The display system tasks displayed with this functionality include: $display, $strobe, $monitor, $write as well as the analogous file I/O tasks that write to STDOUT, such as $fwrite or $fdisplay.

- -do "<command_string>" | <macro_file_name>

  (optional) 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. Multiple commands should be separated by semi-colons (;).

- -donotcollapsepartiallydriven

  (optional) Prevents the collapse of partially driven and undriven output ports during optimization. Prevents incorrect values that can occur when collapsed.

- +dumpports+collapse | +dumpports+nocollapse

  (optional) Determines whether vectors (VCD id entries) in dumpports output are collapsed or not. The default behavior is collapsed, and can be changed by setting the DumpportsCollapse variable in the *modelsim.ini* file.

- +dumpports+direction

  (optional) Modifies the format of extended VCD files to contain direction information.

- +dumpports+no_strength_range

  (optional) Ignores strength ranges when resolving driver values for an extended VCD file. This argument is an extension to the IEEE 1364 specification. Refer to “Resolving Values” for additional information.

- +dumpports+unique

  (optional) Generates unique VCD variable names for ports in a VCD file even if those ports are connected to the same collapsed net.

- -enumfirstinit

  (optional) Initializes enum variables in SystemVerilog using the leftmost value as the default. You must also use the argument with the vlog command in order to implement this initialization behavior. Specify the EnumBaseInit variable as 0 in the *modelsim.ini* file to set this as a permanent default.

- -error <msg_number>[,<msg_number>,…]

  (optional) Changes the severity level of the specified message(s) to "error.” Edit the error variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- -errorfile <filename>

  (optional) Designates an alternative file for recording error messages. An alternate file may also be specified by the ErrorFile modelsim.ini variable. By default, error messages are output to the file specified by the TranscriptFile variable in the *modelsim.ini* file (refer to “Creating a Transcript File”).

- -f <filename>

  (optional) Specifies a file with more **vsim** command arguments. Allows complex argument strings to be reused without retyping.

  Refer to the section "Argument Files" for more information.

- -fatal <msg_number>[,<msg_number>,…]

  (optional) Changes the severity level of the specified message(s) to "fatal." Edit the fatal variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- -g <Name>=<Value> …

  (optional) Assigns a value to all specified VHDL generics and Verilog parameters that have not received explicit values in generic maps, instantiations, or via defparams (such as top-level generics/parameters and generics/parameters that would otherwise receive their default values).

  Note there is a space between **-g** and <Name>=<Value> however, no spaces are allowed in the specification, unless enclosed in quotes when specifying a string value. Multiple **-g** options are allowed, one for each generic/parameter, specified as a space separated list.

<Name> — Name of a generic/parameter, exactly as it appears in the VHDL source (case is ignored) or Verilog source. Name may be prefixed with a relative or absolute hierarchical path to select generics in an instance-specific manner. For example, specifying -g/top/u1/tpd=20ns on the command line would affect only the *tpd* generic on the */top/u1* instance, assigning it a value of 20ns. Specifying -gu1/tpd=20ns affects the *tpd* generic on all instances named *u1*. Specifying-gtpd=20ns affects all generics named *tpd*.

<Value> — Specifies an appropriate value for the declared data type of a VHDL generic or any legal value for a Verilog parameter. Make sure the value you specify for a VHDL generic is appropriate for VHDL declared data types. Integers are treated as signed values. For example, -gp=-10 overwrites the parameter p with the signed value of -10.

If more than one -g option selects a given generic the most explicit specification takes precedence. For example,

```
vsim -g/top/ram/u1/tpd_hl=10ns -gtpd_hl=15ns top
```

This command sets *tpd_hl* to 10ns for the */top/ram/u1* instance. However, all other *tpd_hl* generics on other instances will be set to 15ns.

Limitation: In general, generics/parameters of composite type (arrays and records) cannot be set from the command line. However, you can set string arrays, std_logic vectors, and bit vectors if they can be set using a quoted string. For example,

```
-gstrgen="This is a string"
-gslv="01001110"
```

The quotation marks (" ") must make it into vsim as part of the string because the type of the value must be determinable outside of any context. Therefore, when entering this command from a shell, put single quotes (‘ ‘) around the string. For example:

```
-gstrgen='"This is a string"'
```

If working within the ModelSim GUI, you would enter the command as follows:

```
{-gstrgen="This is a string"}
```

You can also enclose the value escaped quotes (\"), for example:

```
-gstrgen=\"This is a string\"
```

- -G<Name>=<Value> …

  (optional) Same as **-g** (see above) except that it will also override generics/parameters that received explicit values in generic maps, instantiations, or from defparams.

  Note there is a space between **-G** and <Name>=<Value> however, no spaces are allowed in the specification, unless enclosed in quotes when specifying a string value. This argument is the only way for you to alter the generic/parameter, such as its length, (other than its value) after the design has been loaded.

  <Name> — Name of a generic/parameter, exactly as it appears in the VHDL source (case is ignored) or Verilog source. Name may be prefixed with a relative or absolute

hierarchical path to select generics in an instance-specific manner. For example, specifying -G/top/u1/tpd=20ns on the command line would affect only the *tpd* generic on the */top/u1* instance, assigning it a value of 20ns. Specifying -Gu1/tpd=20ns affects the *tpd* generic on all instances named *u1*. Specifying -Gtpd=20ns affects all generics named *tpd*.

<Value> — Specifies an appropriate value for the declared data type of a VHDL generic or any legal value for a Verilog parameter. Make sure the value you specify for a VHDL generic is appropriate for VHDL declared data types. Integers are treated as signed values. For example, -Gp=-10 overwrites the parameter p with the signed value of -10.

- -gblso <filename>

  (optional) On UNIX platforms, loads PLI/FLI shared objects with global symbol visibility. Essentially all data and functions are exported from the specified shared object and are available to be referenced and used by other shared objects. You can also specify this argument with the GlobalSharedObjectsList variable in the *modelsim.ini* file.

- -geometry <geometry_spec>

  (optional) Specifies the size and location of the main window. Where <geometry_spec> is of the form:

  ```
  WxH+X+Y
  ```

- -gui

  (optional) Starts the ModelSim GUI without loading a design and redirects the standard output (stdout) to the GUI Transcript window.

- -help

  (optional) Sends the arguments and syntax for **vsim** to the transcript.

- -i

  (optional) Specifies that the simulator be run in interactive mode.

- +initregNBA

  (optional) Specifies that +initreg settings applied to registers of sequential UDPs should be non-blocking. This is useful when continuous assignments overwrite register initialization.

- -installcolormap

  (optional) 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.

- -keeploaded

  (optional) Prevents the simulator from unloading/reloading any FLI/PLI/VPI shared libraries when it restarts or loads a new design. 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

  (optional) Prevents the simulator from unloading/reloading any FLI/PLI/VPI shared libraries during a restart. 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

  (optional) For use with foreign programs. Instructs the simulator to not redirect the stdout stream to the Main window.

- -l <filename>

  (optional) Saves the contents of the Transcript window to <filename>. Default is taken from the TranscriptFile variable (initially set to *transcript*) in the *modelsim.ini*. You can also specify "stdout" or "stderr" as <filename>.

- -L <library_name> …

  (optional) Specifies the library to search for design units instantiated from Verilog and for VHDL default component binding. Refer to "Library Usage" for more information. If multiple libraries are specified, each must be preceded by the **-L** option. Libraries are searched in the order in which they appear on the command line.

- -Lf <library_name> …

  (optional) Same as **-L** but libraries are searched before 'uselib directives. Refer to "Library Usage" for more information.

- -lib <libname>

  (optional) Specifies the default working library where **vsim** will look for the design unit(s). Default is "work".

- -msglimit <msg_number>[,<msg_number>,…]

  (optional) Limits the number of iterations of the specified message(s) to five then suppresses all new instances. Refer to "Suppressing VSIM Warning Messages" for more information.

  <msg_number>[,<msg_number>,…] — Specifies the message number(s) to limit to five iterations. Multiple messages are specified as a comma-separated list.

- -msgmode both | tran | wlf

  (optional) Specifies the location(s) for the simulator to output elaboration and runtime messages.

  both — outputs messages to both the transcript and the WLF file.

  tran — outputs messages only to the transcript, therefore they are not available in the Message Viewer. Default behavior.

wlf — outputs messages only to the WLF file/Message Viewer , therefore they are not available in the transcript.

Refer to the section "Message Viewer Window" in the User's Manual for more information.

- -modelsimini <ini_filepath>

  (optional) Loads an alternate initialization file that replaces the current initialization file. Overrides the file path specified in the MODELSIM environment variable. Specifies either an absolute or relative path to the initialization file. On Windows systems the path separator should be a forward slash (/).

- -multisource_delay min | max | latest

  (optional) Controls the handling of multiple PORT or INTERCONNECT constructs that terminate at the same port. By default, the Module Input Port Delay (MIPD) is set to the max value encountered in the SDF file. Alternatively, you 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

  (optional) Enables multisource interconnect delay with pulse handling and transport delay behavior. Works for both Verilog and VITAL cells.

  Use this argument when you have interconnect data in your SDF file and you want the delay on each interconnect path modeled independently. Pulse handling is configured using the **+pulse_int_e** and **+pulse_int_r** switches (described below).

  The **+multisource_int_delays** argument cannot be used if you compiled using the **-novital** argument to **vcom**. The -novital argument instructs vcom to implement VITAL functionality using VHDL code instead of accelerated code, and multisource interconnect delays cannot be implemented purely within VHDL.

- -name <name>

  (optional) Specifies the application name used by the interpreter for send commands. This does not affect the title of the window.

- -noautoldlibpath

  (optional) Disables the default internal setting of LD_LIBRARY_PATH, enabling you to set it yourself. Use this argument to make sure that LD_LIBRARY_PATH is not set automatically while you are using the GUI,

- -nocapacity

  (optional) Disables the display of both coarse-grain and fine-grain analysis of memory capacity.

- -nocompress

  (optional) Causes VSIM to create uncompressed checkpoint files. This option may also be specified with the CheckpointCompressMode variable in the *modelsim.ini* file.

- -noimmedca

  (optional) Causes Verilog event ordering to occur without enforced prioritization—continuous assignments and primitives are not run before other normal priority processes scheduled in the same iteration. Use this argument to prevent the default event ordering where continuous assignments and primitives are run with “immediate priority.” You may also set even ordering with the ImmediateContinuousAssign variable in the *modelsim.ini* file.

- +no_notifier

  (optional) Disables the toggling of the notifier register argument of all timing check system tasks. By default, the notifier is toggled when there is a timing check violation, and the notifier usually causes a UDP to propagate an X. This argument suppresses X propagation in both Verilog and VITAL for the entire design.

- -noschematic

  (optional) Used with vsim -debugdb to generate a debug database for the Dataflow window only. Will not remove schematic information that has already been generated by with **vopt -debugdb**.

- +nospecify

  (optional) Disables specify path delays and timing checks in Verilog.

- -nostdout

  (optional) Directs all output to the transcript only when in command line and batch mode. Prevents duplication of I/O between the shell and the transcript file. Has no affect on interactive GUI mode.

- +no_tchk_msg

  (optional) Disables error messages generated when timing checks are violated. For Verilog, it disables messages issued by timing check system tasks. For VITAL, it overrides the MsgOn arguments and generics.

  Notifier registers are still toggled and may result in the propagation of Xs for timing check violations.

- -note <msg_number>[,<msg_number>,…]

  (optional) Changes the severity level of the specified message(s) to "note." Edit the note variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- +notimingchecks | +ntcnotchks

  (optional) Disables Verilog timing checks. (This option sets the generic TimingChecksOn to FALSE for all VHDL Vital models with the Vital_level0 or Vital_level1 attribute. Generics with the name TimingChecksOn on non-VITAL models are unaffected.) 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.

Additionally, +ntcnotchks maintains the delay net delays necessitated by negative timing check limits. For this reason when using +ntcnotchks it is necessary to SDF annotate all timing check values.

- -nowiremodelforce

  (optional) Restores the force command to previous usage (prior to version 10.0b) where an input port cannot be forced directly if it is mapped at a higher level in VHDL and mixed models. Signals must be forced at the top of the hierarchy connected to the input port.

- -pduignore[=<instpath>]

  Ignore Preoptimized Design Unit (black-box). If <instpath> is not specified all PDUs found in compiled libraries will be ignored. Otherwise the PDU specified by <instpath> will be ignored. This option may be specified multiple times with different <instpath>s. Equivalet to the deprecated “-ignore_bbox” option.

- -permissive

  (optional) Allows messages in the LRM group of error messages to be downgraded to a warning.

- -plicompatdefault [latest | 2005 | 2001]

  (optional) Specifies the VPI object model behavior within vsim. This switch applies globally, not to individual libraries.

  latest — This is equivalent to the "2009" argument. This is the default behavior if you do not specify this switch or if you specify the switch without an argument.

  2009 — Instructs vsim to use the object models as defined in IEEE Std P1800-2009 (unapproved draft standard). You can also use "09" as an alias.

  2005 — Instructs vsim to use the object models as defined in IEEE Std 1800-2005 and IEEE Std 1364-2005. You can also use "05" as an alias.

  2001 — Instructs vsim to use the object models as defined in IEEE Std 1364-2001. When you specify this argument, SystemVerilog objects will not be accessible. You can also use "01" as an alias.

  You can also control this behavior with the PliCompatDefault variable in the modelsim.ini file, where the -plicompatdefault argument will override the PliCompatDefault variable.

  You should note that there are a few cases where the 2005 VPI object model is incompatible with the 2001 model, which is inherent in the specifications.

  Refer to the appendix "Verilog Interfaces to C" in the User’s Manual for more information.

- -printsimstats

  (optional) Prints the output of the **simstats** command to the transcript at the end of simulation before exiting. Edit the PrintSimStats variable in the *modelsim.ini* file to set the simulation to print the simstats data by default.

- +pulse_int_e/<percent>

  (optional) Controls how pulses are propagated through interconnect delays, where <percent> is a number between 0 and 100 that specifies the error limit as a percentage of the interconnect delay. Used in conjunction with **+multisource_int_delays** (see above). This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog.

  A pulse greater than or equal to the error limit propagates to the output in transport mode (transport mode allows multiple pending transitions on an output). A pulse less than the error limit and greater than or equal to the rejection limit (see **+pulse_int_r/<percent>** below) propagates to the output as an X. If 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>

  (optional) Controls how pulses are propagated through interconnect delays, where <percent> is a number between 0 and 100 that specifies the rejection limit as a percentage of the interconnect delay. This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source 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

  (optional) Disable 'Loading' messages during batch-mode simulation.

- -runinit

  (optional) Initializes non-trivial static SystemVerilog variables, for example expressions involving other variables and function calls, before displaying the simulation prompt.

- +sdf_iopath_to_prim_ok

  (optional) Prevents **vsim** from issuing an error when it cannot locate specify path delays to annotate. If you specify this argument, IOPATH statements are annotated to the primitive driving the destination port if a corresponding specify path is not found. Refer to "SDF to Verilog Construct Matching" for additional information.

- -sdfmin | -sdftyp | -sdfmax[@<delayScale>] [<instance>=]<sdf_filename>

  (optional) Annotates VITAL or Verilog cells in the specified SDF file (a Standard Delay Format file) with minimum, typical, or maximum timing.

  @<delayScale> — scales all values by the specified value. For example, if you specify -sdfmax@1.5, all maximum values in the SDF file are scaled to 150% of their original value.

<instance>= — specifies a specific instance for the associated SDF file. Use this when not performing backannotation at the top level.

<sdf_filename> — specifies the file containing the SDF information.

- -sdfminr | -sdftypr | -sdfmaxr[@<delayScale>] [<instance>=]<sdf_filename>

(optional) Specifies when an instance of a Preoptimized Design Unit (vopt -pdu, formerly -bbox) with an associated default SDF file is to be re-annotated with minimum, typical, or maximum timing from the specified SDF file.

@<delayScale> — scales all values by the specified value. For example, if you specify -sdfmax@1.5, all maximum values in the SDF file are scaled to 150% of their original value.

<instance>= — specifies a specific instance for the associated SDF file. Use this when not performing back-annotation at the top level.

<sdf_filename> — specifies the file containing the SDF information.

Note

The simulator assumes that the instance/timing object hierarchy in the new SDF file is compatible with the SDF file specified together with vopt -pdu (black-boxing).

The following is a simple usage flow:

Assume module **top** contains three instances (u1, u2, and u3) of a Preoptimized Design Unit named **pduMod**.

```
vlib work
vlog pduMod.v
```

Preoptimize **pduMod** and annotate with *sdf1*.

```
vopt -pdu pduMod -o pduMod_opt -sdfmin pduMod=sdf1
vlog top.v
```

Use the default SDF file *sdf1* for the PDU instance of u1, but override the SDF for u2 and u3.

```
vsim top +sdf_verbose -sdftypr /top/u2=sdf2 -sdfmaxr /top/u3=sdf3
run -all
```

- -sdfmaxerrors <n>

(optional) Controls the number of Verilog SDF missing instance messages to be generated before terminating vsim. <n> is the maximum number of missing instance error messages to be emitted. The default number is 5.

- -sdfnoerror

(optional) Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue.

- -sdfnowarn

  (optional) Disables warnings from the SDF reader. Refer to "VHDL Simulation" for an additional discussion of SDF.

- +sdf_report_unannotated_insts

  (optional) Enables error messages for any un-annotated Verilog instances with specify blocks or VHDL instances with VITAL timing generics that are under regions of SDF annotation.

- +sdf_verbose

  (optional) Turns on the verbose mode during SDF annotation. The Transcript window provides detailed warnings and summaries of the current annotation as well as information including the module name, source file name and line number.

- -suppress <msg_number>[,<msg_number>,…]

  (optional) Prevents the specified message(s) from displaying. You cannot suppress Fatal or Internal messages. Edit the suppress variable in the *modelsim.ini* file to set a permanent default. Refer to "Changing Message Severity Level" for more information.

- -sync

  (optional) Executes all X server commands synchronously, so that errors are reported immediately. Does not apply to Windows platforms.

- -t [<multiplier>]<time_unit>

  (optional) Specifies the simulator time resolution. <time_unit> must be one of the following:

  ```
  fs, ps, ns, us, ms, sec
  ```

  The default is 1ps; the optional <multiplier> may be 1, 10 or 100. Note that there is no space between the multiplier and the unit (for example, 10fs, not 10 fs).

  If you omit the **-t** argument, the default simulator time resolution depends on design type:

  - In a VHDL design—the value specified for the Resolution variable in modelsim.ini is used.
  - In a Verilog design with 'timescale directives—the minimum specified time precision of all directives is used.
  - In a Verilog design with no 'timescale directives—the value specified for the Resolution variable in the modelsim.ini file is used.
  - In a mixed design with VHDL on top—the value specified for the Resolution variable in the modelsim.ini file is used.
  - In a mixed design with Verilog on top—
    - for Verilog modules not under a VHDL instance: the minimum value specified for their 'timescale directives is used.

- for Verilog modules under a VHDL instance: all their ‘timescale directives are ignored (the minimum value for ‘timescale directives in all modules not under a VHDL instance is used).

If there are no ‘timescale directives in the design, the value specified for the Resolution variable in modelsim.ini is used.

Tip: After you have started a simulation, you can view the current simulator resolution by using the report command as follows:

```
report simulator state
```

- -tab <tabfile>

  (optional) Specifies the location of a Synopsys VCS “tab” file (.tab), which the simulator uses to automate the registration of PLI functions in the design.

  <tabfile> — The location of a *.tab* file contains information about PLI functions. The tool expects the *.tab* file to be based on Synopsys VCS version 7.2 syntax. Because the format for this file is non-standard, changes to the format are outside of the control of Mentor Graphics.

- -tag <string>

  (optional) Specifies a string tag to append to foreign trace filenames. Used with the **-trace_foreign <int>** option. Used when running multiple traces in the same directory.

- -title <title>

  (optional) Specifies the title to appear for the ModelSim Main window. If 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>

  (optional) Creates two kinds of foreign interface traces: a log of what functions were called, with the value of the arguments, and the results returned; and a set of C-language files to replay what the foreign interface side did.

  The purpose of the logfile is to aid the debugging of your PLI/VPI code. The primary purpose of the replay facility is to send the replay file to MTI support for debugging co-simulation problems, or debugging problems for which it is impractical to send the PLI/VPI code.

- -usenonstdcoveragesavesysf

  (optional) Replaces implementation of the built-in, IEEE 1800 compliant system function with the non-standard variant, and thus affects all calls to $coverage_save(). The action of this switch is global.

- -uvmcontrol={<args>}

  (optional) Controls UVM-Aware debug features. These features work with either a standard Accelera-released open source toolkit or the pre-compiled UVM library package in ModelSim.

  {<args>}

  You must specify at least one argument. You can enable or disable some arguments by prefixing the argument with a dash (-). Refer to the argument descriptions for more information.

  all — Enables all UVM-Aware functionality and debug options except disable and verbose. You must specify verbose separately.

  certe — Enables the integration of the elaborated design in the Certe tool. Disables Certe features when specified as -certe.

  disable — Prevents the UVM-Aware debug package from being loaded. Changes the results of randomized values in the simulator.

  msglog — Enables messages logged in UVM to be integrated into the Message Viewer. You must also enable wlf message logging by specifying tran or wlf with vsim -msgmode. Disables message logging when specified as -msglog

  none — Turns off all UVM-Aware debug features. Useful when multiple -uvmcontrol options are specified in a separate script, makefile or alias and you want to be sure all UVM debug features are turned off.

  struct — (default) Enables UVM component instances to appear in the Structure window. UVM instances appear under “uvm_root” in the Structure window. Disables Structure window support when specified as -struct.

  trlog — Enables or disables UVM transaction logging. Logs UVM transactions for viewing in the Wave window. Disables transaction logging when specified as -trlog.

  verbose — Sends UVM debug package information to the transcript. Does not affect functionality. Must be specified separately.

  Arguments may be specified as multiple instances of -uvmcontrol. Multiple arguments are specified as a comma separated list without spaces. For example,

  ```
  vsim -uvmcontrol=all,-trlog
  ```

  enables all UVM features except UVM transaction logging. Where arguments are in conflict, the last argument will override earlier arguments and a warning is issued.

  You can also control UVM-Aware debugging with the UVMControl modelsim.ini variable.

- -vcdstim [<instance>=]<filename>

  (optional) Specifies a VCD file from which to re-simulate the design. The VCD file must have been created in a previous ModelSim simulation using the vcd dumpports command. Refer to “Using Extended VCD as Stimulus” for more information.

- -version

  (optional) Returns the version of the simulator as used by the licensing tools.

- -view [<alias_name>=]<WLF_filename>

  (optional) Specifies a wave log format (WLF) file for **vsim** to read. Allows you to use **vsim** to view the results from an open simulation (*vsim.wlf*) or an earlier saved simulation. The Structure, Objects, Wave, and List windows can be opened to look at the results stored in the WLF file (other ModelSim windows will not show any information when you are viewing a dataset).

  <alias_name> — Specifies an alias for **<WLF_file_name>** where the default is to use the prefix of the WLF_filename. Wildcard characters are allowed.

  <WLF_file_name> — Specifies the pathname of a saved WLF file.

  See additional discussion in the Examples.

- -viewcov [<dataset_name>=]<UCDB_filename>

  (required for coverage view mode) Invokes vsim in the coverage view mode to display UCDB data.

- -visual <visual>

  (optional) Specifies the visual to use for the window. Does not apply to Windows platforms.

  Where <visual> may be:

  <class> <depth> — One of the following:

  {directcolor | grayscale | greyscale | pseudocolor | staticcolor | staticgray | staticgrey | truecolor}

  followed by:

  <depth> — Specifies how many bits per pixel are needed for the visual.

  default — Instructs the tool to use the default visual for the screen

  <number> — Specifies a visual X identifier.

  best <depth> — Instructs the tool to choose the best possible visual for the specified <depth>, where:

  <depth> — Specifies how many bits per pixel are needed for the visual.

- -warning <msg_number>[,<msg_number>,…]

  (optional) Changes the severity level of the specified message(s) to "warning." Edit the warning variable in the *modelsim.ini* file to set a permanent default. Refer to “Changing Message Severity Level” for more information.

- -wlf <file_name>

  (optional) Specifies the name of the wave log format (WLF) file to create. The default file name is *vsim.wlf*. This option may also be specified with the WLFFilename variable in the *modelsim.ini* file.

- -wlfcachesize <n>

  (optional) Specifies the size in megabytes of the WLF reader cache. By default the cache size is set to zero. WLF reader caching caches blocks of the WLF file to reduce redundant file I/O. This should have significant benefit in slow network environments. This option may also be specified with the WLFCacheSize variable in the *modelsim.ini* file.

- -wlfcollapsedelta

  (default) Instructs ModelSim to record values in the WLF file only at the end of each simulator delta step. Any sub-delta values are ignored. May dramatically reduce WLF file size. This option may also be specified with the WLFCollapseMode variable in the *modelsim.ini* file.

- -wlfcollapsetime

  (optional) Instructs ModelSim to record values in the WLF file only at the end of each simulator time step. Any delta or sub-delta values are ignored. May dramatically reduce WLF file size. This option may also be specified with the WLFCollapseMode variable in the *modelsim.ini* file.

- -nowlfcollapse

  (optional) Instructs ModelSim to preserve all events for each logged signal and their event order to the WLF file. May result in relatively larger WLF files. This option may also be specified with the WLFCollapseMode variable in the *modelsim.ini* file.

- -wlfcompress

  (default) Creates compressed WLF files. Use **-wlfnocompress** to turn off compression. This option may also be specified with the WLFCompress variable in the *modelsim.ini* file.

- -nowlfcompress

  (optional) Causes **vsim** to create uncompressed WLF files. WLF files are compressed by default in order to reduce file size. This may slow simulation speed by one to two percent. You may want to disable compression to speed up simulation or if you are experiencing problems with faulty data in the resulting WLF file. This option may also be specified with the WLFCompress variable in the *modelsim.ini* file.

- -wlfdeleteonquit

  (optional) Deletes the current simulation WLF file (*vsim.wlf*) automatically when the simulator exits. This option may also be specified with the WLFDeleteOnQuit variable in the *modelsim.ini* file.

- -nowlfdeleteonquit

  (default) Preserves the current simulation WLF file (vsim.wlf) when the simulator exits. This option may also be specified with the WLFDeleteOnQuit variable in the *modelsim.ini* file.

- -wlflock

  (optional) Locks a WLF file. An invocation of ModelSim will not overwrite a WLF file that is being written by a different invocation.

- -nowlflock

  (optional) Disables WLF file locking. This will prevent vsim from checking whether a WLF file is locked prior to opening it as well as preventing vsim from attempting to lock a WLF once it has been opened.

- -wlfopt

  (default, optional) Optimizes the WLF file. Enables faster display of waveforms in the Wave window when the display is zoomed out to display a larger time range. This option may also be specified with the WLFOptimize variable in the *modelsim.ini* file.

- -nowlfopt

  (optional) Disables optimization of waveform display in the Wave window. This option may also be specified with the WLFOptimize variable in the *modelsim.ini* file.

- -wlfsimcachesize <n>

  (optional) Specifies the size in megabytes of the WLF reader cache for the current simulation dataset only. By default the cache size is set to zero. This makes it easier to set different sizes for the WLF reader cache used during simulation and those used during postsimulation debug. WLF reader caching caches blocks of the WLF file to reduce redundant file I/O. If neither the -wlfsimcachesize switch nor the WLFSimCacheSize *modelsim.ini* variable are specified, the -wlfcachesize switch or the WLFCacheSize *modelsim.ini* variable settings will be used.

- -wlfslim <size>

  (optional) Specifies a size restriction for the event portion of the WLF file.

  size — an integer, in megabytes, where the default is 0, which implies an unlimited size.

Note

Note that a WLF file contains event, header, and symbol portions. The size restriction is placed on the event portion only. Consequently, the resulting file will be larger than the specified size.

If used in conjunction with **-wlftlim**, the more restrictive of the limits takes precedence.

This option may also be specified with the WLFSizeLimit variable in the *modelsim.ini* file. (See Limiting the WLF File Size.)

- -wlftlim <duration>

  (optional) Specifies the duration of simulation time for WLF file recording. The default is infinite time (0). The <duration> is an integer of simulation time at the current resolution; you can optionally specify the resolution if you place curly braces around the specification. For example,

  ```
  {5000 ns}
  ```

  sets the duration at 5000 nanoseconds regardless of the current simulator resolution.

The time range begins at the current simulation time and moves back in simulation time for the specified duration. For example,

```
vsim -wlftlim 5000
```

writes at most the last 5000ns of the current simulation to the WLF file (the current simulation resolution in this case is ns).

If used in conjunction with **-wlfslim**, the more restrictive of the limits will take effect.

This option may also be specified with the WLFTimeLimit variable in the *modelsim.ini* file.

The **-wlfslim** and **-wlftlim** switches were designed to help users limit WLF file sizes for long or heavily logged simulations. When small values are used for these switches, the values may be overridden by the internal granularity limits of the WLF file format. (See Limiting the WLF File Size.)

Arguments, VHDL

- -absentisempty

  (optional) Causes VHDL files opened for read that target non-existent files to be treated as empty, rather than ModelSim issuing fatal error messages.

- -nocollapse

  (optional) Disables the optimization of internal port map connections.

- -nofileshare

  (optional) Turns off file descriptor sharing. By default ModelSim shares a file descriptor for all VHDL files opened for write or append that have identical names.

- -noglitch

  (optional) Disables VITAL glitch generation.

  Refer to "VHDL Simulation" for additional discussion of VITAL.

- +no_glitch_msg

  (optional) Disable VITAL glitch error messages.

- -novhdlvariablelogging

  (optional) This switch turns off the ability to log recursively or add process variables to the Wave or List windows. Refer to -vhdlvariable logging and VhdlVariableLogging modelsim.ini variable for more information.

- -std_input <filename>

  (optional) Specifies the file to use for the VHDL TextIO STD_INPUT file.

- -std_output <filename>

  (optional) Specifies the file to use for the VHDL TextIO STD_OUTPUT file.

- -strictvital

  (optional) Specifies to exactly match the VITAL package ordering for messages and delta cycles. Useful for eliminating delta cycle differences caused by optimizations not addressed in the VITAL LRM. Using this argument negatively impacts simulator performance.

- -togglemaxintvalues <int>

  (optional) Specifies the maximum number of VHDL integer values to record for toggle coverage. This limit variable may be changed on a per-signal basis. The default value of <int> is 100 values.

- +transport_int_delays

  (optional) Selects transport mode with pulse control for single-source nets (one interconnect path). By default interconnect delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through interconnect delays.

  This option works for both Verilog and VITAL cells, though the destination of the interconnect must be a Verilog cell. The source may be VITAL or Verilog. This option works independently from **+multisource_int_delays**.

- +transport_path_delays

  (optional) Selects transport mode for path delays. By default, path delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through path delays. Note that this option affects path delays only, and not primitives. Primitives always operate in inertial delay mode.

- +typdelays

  (default) Selects the typical value in min:typ:max expressions. Has no effect if you specified the min:typ:max selection at compile time.

  If you specify the **+mindelays**, **+typdelays**, or **+maxdelays** flag with **vopt**, and specify a different flag with **vsim**, the simulation will be able to use the delay value based upon the flag specified with **vopt**. You must specify **vsim -novopt** to force the simulator to use the delay flag specified with **vsim**.

- -vhdlvariablelogging

  (optional) This switch makes it possible for process variables to be logged recursively or added to the Wave and List windows (process variables can still be logged or added to the Wave and List windows explicitly with or without this switch). For example with this vsim switch, log -r /* will log process variables as long as vopt is specified with +acc=v and the variables are not filtered out by the WildcardFilter (via the "Variable" entry). You can disable this argument with -novhdlvariablelogging. Refer to-vhdlvariable logging and VhdlVariableLogging modelsim.ini variable for more information.

Note

Logging process variables is inherently expensive on simulation performance because of their nature. It is recommended that they not be logged, or added to the Wave and List windows. However, if debugging requires them to be logged, then use of this switch will lessen the performance hit in doing so.

- -vital2.2b

  (optional) Selects SDF mapping for VITAL 2.2b (default is VITAL 2000).

Arguments, Verilog

- +alt_path_delays

  (optional) Configures path delays to operate in inertial mode by default. 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>**).

- -classdebug

  (optional) Enables visibility into class instances for class and UVM debugging. You can also enable visibility into class instances by setting the ClassDebug modelsim.ini variable to 1. Refer to the classinfo command for more information.

- +delayed_timing_checks

  (optional) Causes timing checks to be performed on the delayed versions of input ports (used when there are negative timing check limits). By default, ModelSim automatically detects and applies +delayed_timing_checks to cells with negative timing checks. To turn off this feature, specify +no_autodtc with vsim.

- -dpiforceheader

  (optional) Forces the generation of a DPI header file even if it will be empty of function prototypes.

- -dpiheader

  (optional) Generates a header file that may then be included in C source code for DPI import functions. Simulation quits after header file is generated. Refer to "DPI Use Flow" for additional information.

- -dpilib <libname>

  (optional) Specifies the design library name that contains DPI exports and automatically compiled object files. If the -dpilib switch is not set, **vsim** loads export symbols from all

libraries accessible via vsim options **-L**, **-Lf**, and **-lib**. Multiple occurences of **-dpilib** are supported.

- -dpioutoftheblue 0 | 1 | 2

  (optional) Instructs **vsim** to allow DPI out-of-the-blue calls from C functions. The C functions must not be declared as import tasks or functions.

  0 — Support for DPI out-of-the-blue calls is disabled.

  1 — Support for DPI out-of-the-blue calls is enabled, but debugging support is not available.

  2 — Support for DPI out-of-the-blue calls is enabled with debugging support for a SystemC thread.

  Debugging support for DPI out-of-the-blue calls from a SystemC method requires two **vsim** arguments entered together at the command line: **-dpioutoftheblue 2** and **-scdpidebug**. Refer to **-scdpidebug** for more information.

- -hazards

  (optional) Enables event order hazard checking in Verilog modules (Verilog only). You must also specify this argument when you compile your design with vlog. Refer to "Hazard Detection" for more details.

Note

Using -hazards implicitly enables the -compat argument. As a result, using this argument may affect your simulation results.

- +maxdelays

  (optional) Selects the maximum value in min:typ:max expressions. The default is the typical value. Has no effect if you specified the min:typ:max selection at compile time.

  If you specify the **+mindelays**, **+typdelays**, or **+maxdelays** flag with **vopt**, and specify a different flag with **vsim**, the simulation will be able to use the delay value based upon the flag specified with **vopt**. You must specify **vsim -novopt** to force the simulator to use the delay flag specified with **vsim**.

- +mindelays

  (optional) Selects the minimum value in min:typ:max expressions. The default is the typical value. Has no effect if you specified the min:typ:max selection at compile time.

  If you specify the **+mindelays**, **+typdelays**, or **+maxdelays** flag with **vopt**, and specify a different flag with **vsim**, the simulation will be able to use the delay value based upon the flag specified with **vopt**. You must specify **vsim -novopt** to force the simulator to use the delay flag specified with **vsim**.

- +no_autodtc

  (optional) Turns off auto-detection of optimized cells with negative timing checks and auto-application of +delayed_timing_checks to those cells.

- +no_cancelled_e_msg

  (optional) Disables negative pulse warning messages. By default **vsim** issues a warning and then filters negative pulses on specify path delays. You can drive an X for a negative pulse using **+show_cancelled_e**.

- +no_neg_tchk

  (optional) Disables negative timing check limits by setting them to zero. By default negative timing check limits are enabled. This is 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

  (optional) Disables the toggling of the notifier register argument of all timing check system tasks. By default, the notifier is toggled when there is a timing check violation, and the notifier usually causes a UDP to propagate an X. This argument suppresses X propagation on timing violations for the entire design.

- +no_path_edge

  (optional) Causes ModelSim to ignore the input edge specified in a path delay. The result 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

  (optional) Disables the warning message for specify path pulse errors. A path pulse error occurs when a pulse propagated through a path delay falls between the pulse rejection limit and pulse error limit set with the **+pulse_r** and **+pulse_e** options. A path pulse error results in a warning message, and the pulse is propagated as an X. The **+no_pulse_msg** option disables the warning message, but the X is still propagated.

- -no_risefall_delaynets

  (optional) Disables the rise/fall delay net delay negative timing check algorithm. This argument is provided to return ModelSim to its pre-6.0 behavior where violation regions must overlap in order to find a delay net solution. In 6.0 versions and later, ModelSim uses separate rise/fall delays, so violation regions need not overlap for a delay solution to be found.

- +no_show_cancelled_e

  (optional) 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

  (optional) Disables error messages issued by timing check system tasks when timing check violations occur. Notifier registers are still toggled and may result in the propagation of Xs for timing check violations.

- -nodpiexports

  (optional) Instructs ModelSim to not generate C wrapper code for DPI export task and function routines found at elaboration time. More specifically, the command does not generate the *exportwrapper.so* shared object file.

  For a description on when you should use this argument, refer to the section "Deprecated Legacy DPI Flows" in the User's Manual.

- -noexcludehiz

  (optional) Instructs ModelSim to include truth table rows that contain Hi-Z states in the coverage count. Without this argument, these rows are automatically excluded.

- -noexcludeternary

  (optional) Disables the automatic exclusion of UDB coverage data rows resulting from ternary expressions for the entire design. Normal operation for code coverage is to include rows corresponding to the case where two data inputs are the same, and the select input is a "don't care". To disable this automatic exclusion for a specified design unit only, use "vlog -noexcludeternary <design_unit>" instead.

- +nosdferror

  (optional) Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue.

- +nosdfwarn

  (optional) Disables warnings from the SDF annotator.

- +nospecify

  (optional) Disables specify path delays and timing checks.

- +nowarnBSOB

  (optional) Disables run-time warning messages for bit-selects in initial blocks that are out of bounds.

- +nowarn<CODE | number>

  (optional) Disables warning messages in the category specified by a warning code or number. Warnings that can be disabled include the code name in square brackets in the warning message. For example:

  ```
  ** Warning: (vsim-3017) test.v(2): [TFMPC] - Too few port
  connections. Expected <m>, found <n>.
  ```

  The warning code for this example is TFMPC, and the warning number is 3017. Therefore, this warning message can be disabled with **+nowarnTFMPC** or **+nowarn3017**.

- +ntc_warn

  (optional) Enables warning messages from the negative timing constraint algorithm. By default, these warnings are disabled.

This algorithm attempts to find a set of delays for the timing check delayed net arguments such that all negative limits can be converted to non-negative limits with respect to the delayed nets. If there is no solution for this set of limits, then the algorithm sets one of the negative limits to zero and recalculates the delays. This process is repeated until a solution is found. A warning message is issued for each negative limit set to zero.

- +ntcnotchks

  (optional) Instructs vsim to not simulate timing checks but still consider negative timing check limits for the calculation of delayed input delays.

- -oldvhdlforgennames

  (optional) Enables the use of a previous style of naming in VHDL for … generate statement iteration names in the design hierarchy. The previous style is controlled by the value of the GenerateFormat value. The default behavior is to use the current style names, which is described in "Naming Behavior of VHDL For Generate Blocks" This argument duplicates the function of the OldVhdlForGenNames variable in modelsim.ini and will override the setting of that variable if it specifies the current style.

- -onfinish ask | stop | exit | final

  (optional) Customizes the simulator shutdown behavior when it encounters $finish in the design:

  - **ask** —
    - In batch mode, the simulation exits.
    - In GUI mode, a dialog box pops up and asks for user confirmation on whether to quit the simulation.
  - **stop —** stops simulation and leaves the simulation kernal running
  - **exit** — exits out of the simulation without a prompt
  - **final** — executes all final blocks then exits the simulation

  By default, the simulator exits in batch mode; prompts you in GUI mode. Edit the OnFinish variable in the *modelsim.ini* file to set the default operation of $finish.

- -pli "<object list>"

  (optional) Loads a space-separated list of PLI shared objects. The list must be quoted if it contains more than one object. This is an alternative to specifying PLI objects in the Veriuser entry in the *modelsim.ini* file, refer to modelsim.ini Variables. You can use environment variables as part of the path.

- +<plusarg>

  (optional) Arguments preceded with "+" are accessible by the Verilog PLI routine **mc_scan_plusargs()**.

- +pulse_e/<percent>

  (optional) Controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the error limit as a percentage of the path delay.

  A pulse greater than or equal to the error limit propagates to the output in transport mode (transport mode allows multiple pending transitions on an output). A pulse less than the error limit and greater than or equal to the rejection limit (see **+pulse_r/<percent>**) propagates to the output as an X. If 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

  (optional) Selects the "on detect" style of propagating pulse errors (see **+pulse_e**). A pulse error propagates to the output as an X, and the "on detect" style is to schedule the X immediately, as soon as it has been detected that a pulse error has occurred. "on event" style is the default for propagating pulse errors (see **+pulse_e_style_onevent**).

- +pulse_e_style_onevent

  (optional) Selects the "on event" style of propagating pulse errors (see **+pulse_e**). Default. A pulse error propagates to the output as an X, and the "on event" style is to schedule the X to occur at the same time and for the same duration that the pulse would have occurred if it had propagated through normally.

- +pulse_r/<percent>

  (optional) Controls how pulses are propagated through specify path delays, where <percent> is a number between 0 and 100 that specifies the rejection limit as a percentage of the path delay.

  A pulse less than the rejection limit is suppressed from propagating to the output. If the error limit is not specified by **+pulse_e** then it defaults to the rejection limit.

- +sdf_nocheck_celltype

  (optional) Disables the error check a for mismatch between the CELLTYPE name in the SDF file and the module or primitive name for the CELL instance. It is an error if the names do not match.

- +show_cancelled_e

  (optional) Drives a pulse error state (’X’) for the duration of a negative pulse on a specify path delay. By default ModelSim filters negative pulses.

- -sv_lib <shared_obj>

  (required for use with DPI import libraries) Specifies the name of the DPI shared object with no extension. Refer to “DPI Use Flow” for additional information.

- -sv_liblist <filename>

  (optional) Specifies the name of a bootstrap file containing names of DPI shared objects (libraries) to be loaded. Refer to "DPI File Loading" for format information.

- -sv_root <dirname>

  (optional) Specifies the directory name to be used as the prefix for DPI shared object lookups.

- +transport_int_delays

  (optional) Selects transport mode with pulse control for single-source nets (one interconnect path). By default interconnect delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through interconnect delays.

  This 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**.

- +transport_path_delays

  (optional) Selects transport mode for path delays. By default, path delays operate in inertial mode (pulses smaller than the delay are filtered). In transport mode, narrow pulses are propagated through path delays. Note that this option affects path delays only, and not primitives. Primitives always operate in inertial delay mode.

- +typdelays

  (default) Selects the typical value in min:typ:max expressions. Has no effect if you specified the min:typ:max selection at compile time.

  If you specify the **+mindelays**, **+typdelays**, or **+maxdelays** flag with **vopt**, and specify a different flag with **vsim**, the simulation will be able to use the delay value based upon the flag specified with **vopt**. You must specify **vsim -novopt** to force the simulator to use the delay flag specified with **vsim**.

- -v2k_int_delays

  (optional) Causes interconnect delays to be visible at the load module port per the IEEE 1364-2001 spec. By default ModelSim annotates INTERCONNECT delays in a manner compatible with Verilog-XL. If you have $sdf_annotate() calls in your design that are not getting executed, add the Verilog task $sdf_done() after your last $sdf_annotate() to remove any zero-delay MIPDs that may have been created. May be used in tandem with the **+multisource_int_delays** argument (see above).

Arguments, object

The object arguments may be a [<library_name>].<design_unit>, an *.mpf* file, a *.wlf* file, or a text file. Multiple design units may be specified for Verilog modules and mixed VHDL/Verilog configurations.

- <library_name>.<design_unit>

  (optional) Specifies a library and associated design unit; multiple library/design unit specifications can be made. If no library is specified, the **work** library is used. You cannot

use the wildcard * for this argument. Environment variables can be used. <design_unit> may be one of the following:

<configuration>	Specifies the VHDL configuration to simulate.
<module> …	(optional) Specifies the name of one or more top-level Verilog modules to be simulated.
<entity> [(<architecture>)]	(optional) Specifies the name of the top-level VHDL entity to be simulated. The entity may have an architecture optionally specified; if omitted the last architecture compiled for the specified entity is simulated. An entity is not valid if a configuration is specified.[1]

1. Most UNIX shells require arguments containing () to be single-quoted to prevent special parsing by the shell. See the examples below.

- <MPF_file_name>

  (optional) Opens the specified project.

- <WLF_file_name>

  (optional) Opens the specified dataset. When you open a WLF file using the following command:

  vsim test.wlf

  The default behavior is to not automatically load any signals into the Wave window. You can change this behavior, such that the Wave window contains all signals in the design, by setting the preference PrefWave(OpenLogAutoAddWave) to 1 (true).

- <text_file_name>

  (optional) Opens the specified text file in a Source window.

Examples

- Invoke **vsim** on the entity *cpu* and assigns values to the generic parameters *edge* and *VCC*.

  ```
  vsim -gedge='"low high"' -gVCC=4.75 cpu
  ```

  If working within the ModelSim GUI, you would enter the command as follows:

  ```
  vsim {-gedge="low high"} -gVCC=4.75 cpu
  ```

  Instruct ModelSim to view the results of a previous simulation run stored in the WLF file *sim2.wlf*. The simulation is displayed as a dataset named *test*. Use the **-wlf** option to specify the name of the WLF file to create if you plan to create many files for later viewing.

  ```
  vsim -view test=sim2.wlf
  ```

  For example:

```
vsim -wlf my_design.i01 my_asic structure
vsim -wlf my_design.i02 my_asic structure
```

Annotate instance */top/u1* using the minimum timing from the SDF file *myasic.sdf*.

```
vsim -sdfmin /top/u1=myasic.sdf
```

Use multiple switches to annotate multiple instances:

```
vsim -sdfmin /top/u1=sdf1 -sdfmin /top/u2=sdf2 top
```

- This example searches the libraries *mylib* for *top(only)* and *gatelib* for *cache_set*. If the design units are not found, the search continues to the work library. Specification of the architecture (*only*) is optional.

```
vsim 'mylib.top(only)' gatelib.cache_set
```

- Invoke **vsim** on *test_counter* and run the simulation until a break event, then quit when it encounters a $finish task.

```
vsim -do "set PrefMain(forceQuit) 1; run -all" work.test_counter
```

vsim<info>

The **vsim<info>** commands return information about the current vsim executable.

- vsimAuth

  Returns the authorization level (PE/SE, VHDL/Verilog/PLUS).

- vsimDate

  Returns the date the executable was built, such as "Apr 10 2000".

- vsimId

  Returns the identifying string, such as "ModelSim 6.1".

- vsimVersion

  Returns the version as used by the licensing tools, such as "1999.04".

- vsimVersionString

  Returns the full vsim version string.

This same information can be obtained using the **-version** argument of the vsim command.

vsim_break

Stop (interrupt) the current simulation before it runs to completion. To stop a simulation and then resume it, use this command in conjunction with **run -continue**.

Syntax

vsim_break

Arguments

None.

Example

- Interrupt a simulation, then restart it from the point of interruption.

```
vsim_break
run -continue
```

vsource

This command specifies an alternative file to use for the current source file.

This command is used when the current source file has been moved. The alternative source mapping exists for the current simulation only.

Syntax

vsource [<filename>]

Arguments

- <filename>

  (optional) Specifies a relative or full pathname. If filename is omitted, the source file for the current design context is displayed.

Examples

```
vsource design.vhd
vsource /old/design.vhd
```

wave

A number of commands are available to manipulate and report on the Wave window.

The following tables summarize the available options for manipulating cursors, for zooming, and for adjusting the wave display view in the Wave window:

Table 2-8. Wave Window Commands for Cursor

Cursor Commands	Description
wave cursor active	Sets the active cursor to the specified cursor or, if no cursor is specified, reports the active cursor
wave cursor add	Adds a new cursor at specified time and returns the number of the newly added cursor
wave cursor configure	Sets or reports values for the specified cursor
wave cursor delete	Deletes the specified cursor or, if no cursor is specified, the active cursor
wave cursor see	Positions the wave display such that the specified or active cursor appears at the specified percent from the left edge of the display – 0% is the left edge, 100% is the right edge.
wave cursor time	Moves or reports the time of the specified cursor or, if no cursor is specified, the time of the active cursor

Table 2-9. Wave Window Commands for Expanded Time Display

Display view Commands	Description
wave expand mode	Selects the expanded time display mode: Delta Time, Event Time, or off.
wave expand all	Expands simulation time into delta time steps if Delta Time mode is currently selected (WLFCollapseMode = 1) or into event time steps if Event Time mode is currently selected (WLFCollapseMode = 0) over the full range of the simulation from time 0 to the current time.
wave expand cursor	Expands simulation time into delta time steps if Delta Time mode is currently selected (WLFCollapseMode = 1) or into event time steps if Event Time mode is currently selected (WLFCollapseMode = 0) at the simulation time of the active cursor.
wave expand range	Expands simulation time into delta time steps if Delta Time mode is currently selected (WLFCollapseMode = 1) or into event time steps if Event Time mode is currently selected (WLFCollapseMode = 0) over a time range specified by a start time and an end time.

Table 2-9. Wave Window Commands for Expanded Time Display (cont.)

Display view Commands	Description
wave collapse all	Collapses simulation time over the full range of the simulation from time 0 to the current time.
wave collapse cursor	Collapses simulation time at the time of the active cursor.
wave collapse range	Collapses simulation time over a specific simulation time range.

Table 2-10. Wave Window Commands for Controlling Display

Display view Commands	Description
wave interrupt	Immediately stops wave window drawing
wave refresh	Cleans wave display and redraws waves
wave cursor see	Positions the wave display such that the specified or active cursor appears at the specified percent from the left edge of the display – 0% is the left edge, 100% is the right edge.
wave seetime	Positions the wave display such that the specified time appears at the specified percent from the left edge of the display – 0% is the left edge, 100% is the right edge.

Table 2-11. Wave Window Commands for Zooming

Zooming Commands	Description
wave zoom in	Zoom in the wave display by the specified factor. The default factor is 2.0.
wave zoom out	Zoom out the wave display by the specified factor. The default factor is 2.0.
wave zoom full	Zoom the wave display to show the full simulation time.
wave zoom last	Return to last zoom range.
wave zoom range	Sets left and right edge of wave display to the specified start time and end time. If times are not specified, reports left and right edge times.

Syntax

wave cursor active [-window <win>] [<cursor-num>]

wave cursor add [-window <win>] [-time <time>] [-name <name>] [-lock <0 |1>]

wave cursor configure [<cursor-num>] [-window <win>] [<option> [<value>]]

wave cursor delete [-window <win>] [<cursor-num>]

wave cursor see [-window <win>] [-at <percent>] [<cursor-num>]

wave cursor time [-window <win>] [-time <time>] [<cursor-num>]

wave collapse all [-window <win>]

wave collapse cursor [-window <win>] [<cursor-num>]

wave collapse range [-window <win>] <start-time> <end-time>

wave expand all [-window <win>]

wave expand cursor [-window <win>] [<cursor-num>]

wave expand mode [-window <win>] [off | deltas | events]

wave expand range [-window <win>] <start-time> <end-time>

wave interrupt [-window <win>]

wave refresh [-window <win>]

wave seetime [-window <win>] [-at <percent>] -time <time>

wave zoom in [-window <win>] [<factor>]

wave zoom out [-window <win>] [<factor>]

wave zoom full [-window <win>]

wave zoom last [-window <win>]

wave zoom range [-window <win>] [<start-time> <end-time>]

Arguments

- -at <percent>

  (optional) Positions the display such that the time or cursor is the specified <percent> from the left edge of the wave display.

  <percent> — Any non-negative number where the default is 50. 0 is the left edge of the Wave window and 100 is the right edge.

- <cursor-num>

  (optional) Specifies a cursor number. If not specified, the active cursor is used.

- <factor>

  (optional) A number that specifies how much you want to zoom into or out of the wave display. Default value is 2.0.

- -lock <0 |1>

  (optional) Specify the lock state of the cursor.

  0 — (default) Unlocked

  1 — Locked

- -name <name>

  (optional) Specify the name of the cursor.

  <name> — Any string where the default is "Cursor <n>" where <n> is the cursor number.

- off | deltas | events

  (optional) Specifies the expanded time display mode for the Wave window. Default is off.

- <option> [<value>]

  (optional) Specify a value for the designated option. Currently supported options are -name, -time, and -lock. If no option is specified, current value of all options are reported.

- <start-time> <end-time>

  (optional) start-time and end-time are times that specify an expand, collapse, or zoom range. If neither number is specified, the command returns the current range.

- -time <time>

  (optional) Specifies a cursor time.

  <time> — Any positive integer.

- -window <win>

  (optional) All commands default to the active Wave window unless this argument is used to specify a different Wave window.

  <win> — Specifies the name of a Wave window other than the current active window.

Examples

- Either of these commands creates a zoom range with a start time of 20 ns and an end time of 100 ns.

  ```
  wave zoom range 20ns 100ns
  wave zoom range 20 100
  ```

- Return the name of cursor 2:

  ```
  wave cursor configure 2 -name
  ```

- Name cursor 2, "reference cursor" and return that name with:

  ```
  wave cursor configure 2 -name {reference cursor}
  ```

- Return the values of all wave cursor configure options for cursor 2:

  ```
  wave cursor configure 2
  ```

wave create

This command generates a waveform known only to the GUI. You can then modify the waveform interactively or with the wave edit command and use the results to drive simulation.

Refer to "Generating Stimulus with Waveform Editor" for more information.

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

The following table summarizes the available waveform pattern options:

Command	Description
wave create -pattern clock	Generates a clock waveform. Recommended that you specify an initial value, duty cycle, and clock period for the waveform.
wave create -pattern constant	Generates a waveform with a constant value. It is suggested that you specify a value.
wave create -pattern random	Generates a random waveform based upon a seed value. Specify the type (normal or uniform), an initial value, and a seed value. If you don't specify a seed value, Questa uses a default value of 5.
wave create -pattern repeater	Generates a waveform that repeats. Specify an initial value and pattern that repeats. You can also specify how many times the pattern repeats.
wave create -pattern counter	Generates a waveform from a counting pattern. Specify start and end values, repeat, step count, time period, and type (Binary, Gray, Johnson, OneHot, Range, and ZeroHot).
wave create -pattern none	Creates a placeholder for a waveform. Specify an object name.

Syntax

All waveforms

wave create **[-driver {freeze | deposit | driver | expectedoutput}]** [-initialvalue <value>] [-language {vhdl | verilog}] [-portmode {in | out | inout | internal}] [-range <msb lsb>] [-starttime {<time><unit>}] [-endtime {<time><unit>}] **<object_name>**

Clock patterns

wave create **-pattern clock [-dutycycle <value>] [-period {<time><unit>}] <object_name>**

Constant patterns

wave create **-pattern constant** [-initialvalue <value>] **[-value <value>] <object_name>**

Random patterns

wave create **-pattern random** [-initialvalue <value>] **[-period {<time><unit>}] [-random_type {normal | uniform | poisson | exponential}] [-seed <value>] <object_name>**

Repeater patterns

wave create **-pattern repeater** [-initialvalue <value>] **[-period {<time><unit>}] [-repeat {forever | never | <n>}] [-sequence {<val1>} <val2> …]**

Counter patterns

wave create **-pattern counter** [**-direction {up | down | upthendown | downthenup}**] [-initialvalue <value>] **[-period {<time><unit>}] [-repeat {forever | never | <n>}] [-startvalue <value>] [-endvalue <value>] [-step <value>] [-type {binary | gray | johnson | onehot | range | zerohot}] <object_name>**

No pattern

wave create **-pattern none <object_name>**

Arguments

- -pattern clock | constant | random | repeater | counter | none

  (required) Specifies the waveform pattern. Refer to "Creating Waveforms from Patterns" for a description of the pattern types.

  clock — Specifies a clock pattern.

  constant — Specifies a constant pattern.

  random — Specifies a random pattern.

  repeater — Specifies a repeating pattern.

  counter — Specifies a counting pattern.

  none — Specifies a blank pattern.

- -direction {up | down | upthendown | downthenup}

  (optional, recommended when specifying **-pattern counter**) The direction in which the counter will increment or decrement.

  up — (default) Increment only.

  down — Decrement only.

  upthendown — Increment then decrement.

  downthenup — Decrement then increment.

- -driver {freeze | deposit | driver | expectedoutput}

  (optional) Specifies that the signal is a driver of the specified type. Applies to waveforms created with **-portmode inout** or **-portmode internal**.

- -dutycycle <value>

  (optional, recommended for **-pattern clock**) Specifies the duty cycle of the clock. Expressed as a percentage of the period that the clock is high.

  <value> — Any integer from 0 to 100 where the default is 50.

- -endtime {<time><unit>}

  (optional) The simulation time where the waveform will stop. If omitted, the waveform stops at 1000 simulation time units.

  <time> — Specified as an integer or decimal number.

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

- -endvalue <value>

  (optional, recommended when specifying **-pattern counter**) The end value for the counter. This option applies to patterns specifying **-type Range** only. All other counter patterns start from 0 and go to the maximum value for that particular signal (for example, for a 3-bit signal, the start value will be 000 and the end value will be 111).

  <value> — Value must be appropriate for the type of waveform you are creating.

- -initialvalue <value>

  (optional) The initial value for the waveform. Not applicable to counter patterns.

  <value> — Value must be appropriate for the type of waveform you are creating.

- -language {vhdl | verilog}

  (optional) Controls which language is used for the created wave.

  vhdl — (default) Specifies the VHDL language.

  verilog — Specifies the Verilog language.

- -period {<time><unit>}

  (optional, recommended for all patterns except **-constant**) Specifies the period of the signal.

  <time> — Specified as an integer or decimal number. Current simulation units are the default unless specifying **<unit>**.

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

- -portmode {in | out | inout | internal}

  (optional) The port type for the waveform. Useful for creating signals prior to loading a design.

  in — Ports of type IN. You can also specify “input” as an alias for in.

out — Ports of type OUT. You can also specify “output” as an alias for out.

inout — Ports of type INOUT.

internal — (default) Ports of type INTERNAL.

- -random_type {normal | uniform | poisson | exponential}

  (optional, recommended when specifying **-pattern random**) Specifies the type of algorithm used to generate a random waveform pattern.

  normal — Normal or Gaussian distribution of waveform events.

  uniform — (default) Uniform distribution of waveform events.

  poisson — Poisson distribution of waveform events.

  exponential — Exponential distribution of waveform events.

- -range <msb lsb>

  (optional) Identifies bit significance in a counter pattern.

  msb lsb — Most significant bit and least significant bit. Both must be specified.

- -repeat {forever | never | <n>}

  (optional, recommended when specifying **-pattern repeater** or **-pattern counter**) Controls duration of pattern repetition.

  forever — Repeat the pattern for as long as the simulation runs.

  never — Never repeat the pattern during simulation.

  <n> — Repeat the pattern <n> number of times where <n> is any positive integer.

- -seed <value>

  (optional, recommended when specifying **-pattern random**) Specifies a seed value for a randomly generated waveform.

  <value> — Any non-negative integer where the default is 5.

- -sequence {<val1>} <val2> …

  (optional, recommended when specifying **pattern -repeater**) The set of values that you want repeated.

  <val1> — Value must be appropriate for the type of waveform you are creating. Multiple values are entered as a space separated list and enclosed in curly braces ({}).

- -starttime {<time><unit>}

  (optional) The simulation time at which the waveform should start. If omitted, the waveform starts at 0 simulation time units.

  <time> — Specified as an integer or decimal number.

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

- -startvalue <value>

  (required when specifying **-pattern counter**) The initial value of the counter. This option applies to patterns specifying **-type Range** only. All other counter patterns start from 0 and go to the maximum value for that particular signal (e.g., for a 3-bit signal, the start value will be 000 and the end value will be 111).

  <value> — Value must be appropriate for the type of waveform you are creating.

- -step <value>

  (optional, recommended when specifying **-pattern counter**) The step by which the counter is incremented/decremented.

  <value> — Value must be appropriate for the type of waveform you are creating.

- -type {binary | gray | johnson | onehot | range | zerohot}

  (optional) Specifies a counter format.

  binary — Specifies a binary counter.

  gray — Specifies a binary counter where two successive values differ in only one bit. Also known as a reflected binary counter.

  johnson — Specifies a twisted ring or Johnson counter.

  onehot — Specifies a shift counter where only one bit at a time is set to “on” (1).

  range — (default) Specifies a binary counter where the values range between **-startvalue** and **-endvalue**

  zerohot — Specifies a shift counter where only one bit at a time is set to “off” (0).

- -value <value>

  (optional, recommended when specifying **-pattern constant**) Specifies a value for the constant pattern.

  <value> — Value must be appropriate for the type of waveform you are creating.

- <object_name>

  (required) User specified name for the waveform. Must be the final argument.

Examples

- Create a clock signal with the following default values:

  ```
  wave create -pattern clock -period 100 -dutycycle 50 -starttime 0 -endtime 1000
  -initialvalue 0 /counter/clk
  ```

- Create a constant 8-bit signal vector from 0 to 1000 ns with a value of 1111 and a drive type of freeze.

  ```
  wave create -driver freeze -pattern constant -value 1111 -range 7 0 -starttime 0ns
  -endtime 1000ns sim:/andm/v_cont2
  ```

Related Topics

- wave edit
- wave modify
- "Generating Stimulus with Waveform Editor"
- "Creating Waveforms from Patterns"

wave edit

This command modifies waveforms created with the **wave create** command.

Some arguments to this command are order-dependent. Please read through the argument descriptions for more information. d

The following table summarizes the available editing options:

Command	Description
wave edit cut	Cut part of a waveform to the clipboard
wave edit copy	Copy part of a waveform to the clipboard
wave edit paste	Paste the waveform from the clipboard
wave edit invert	Vertically flip part of a waveform
wave edit mirror	Mirror part of a waveform
wave edit insert_pulse	Insert a new edge on a waveform; doesn't affect waveform duration
wave edit delete	Delete an edge from a waveform; doesn't affect waveform duration
wave edit stretch	Move an edge by stretching the waveform
wave edit move	Move an edge without moving other edges
wave edit change_value	Change the value of part of a waveform
wave edit extend	Extend all waves
wave edit driveType	Change the driver type
wave edit undo	Undo an edit
wave edit redo	Redo a previously undone edit

Syntax

wave edit **{cut | copy | paste | invert | mirror} -end {<time><unit>} -start {<time><unit>} <object_name>**

wave edit insert_pulse [-duration {<time><unit>}] **-start {<time><unit>} <object_name>**

wave edit delete **-time {<time><unit>} <object_name>**

wave edit stretch | move **{-backward {<time><unit>} | -forward {<time><unit>}} -time {<time><unit>} <object_name>**

wave edit change_value **-end {<time><unit>} -start {<time><unit>} <value> <object_name>**

wave edit extend **-extend to | by -time {<time><unit>}**

wave edit driveType **-driver freeze | deposit | driver | expectedoutput -end {<time><unit>} -start {<time><unit>}**

wave edit undo **<number>**

wave edit redo **<number>**

Arguments

- -backward {<time><unit>}

  (required if -forward <time> isn't specified) The amount to stretch or move the edge backwards in simulation time.

  <time> — Specified as an integer or decimal number.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- cut | copy | paste | invert | mirror

  (required) Specifies the type of edit to perform.

  cut — Deletes the specified portion of the waveform.

  copy — Saves a copy of the specified portion of the waveform.

  paste — Inserts the contents of the clipboard into the specified portion of the waveform.

  invert — Flips the specified portion of the waveform vertically.

  mirror — Flips the specified portion of the waveform horizontally.

- -driver freeze | deposit | driver | expectedoutput

  (required) Specifies the type of driver to which you want the specified section of the waveform changed. Applies to signals of type inout or internal.

- -duration {<time><unit>}

  (optional) The length of the pulse.

  <time> — Specified as an integer or decimal number where the default is 10 time units.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- -end {<time><unit>}

  (required unless specifying paste) The end of the section of waveform to perform the editing operation upon, denoted by a simulation time.

  <time> — Specified as an integer or decimal number.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- -extend to | by

  (required) Specifies the format for extending waves.

to — Extends the wave to the time specified by -time <time>.

by — Extends the wave by the amount of time specified by -time <time>.

- -forward {<time><unit>}

  (required if -backward <time> is not specified) The amount to stretch or move the edge forwards in simulation time.

  <time> — Specified as an integer or decimal number.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- <number>

  (optional) The number of editing operations to undo or redo. If omitted, only one editing operation is undone or redone.

- <object_name>

  (required) The pathname of the waveform to edit. Must be specified as the last argument to **wave edit**.

- -start {<time><unit>}

  (required) The beginning of the section of waveform to perform the editing operation upon, denoted by a simulation time.

  <time> — Specified as an integer or decimal number.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- -time {<time><unit>}

  (required) The amount of time to extend or stretch waves.

  <time> — Specified as an integer or decimal number.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- <value>

  (required) The new value. Must match the type of the <object_name>.

Related Topics

- wave create
- "Generating Stimulus with Waveform Editor"

wave export

This command creates a stimulus file from waveforms created with the wave create command.

Syntax

wave export **-designunit <name> -endtime {<time><unit>} -<time> -file <name> {-format force | vcd | vhdl | verilog} -starttime <time>**

Arguments

- -designunit <name>

  (required) Specifies a design unit for which you want to export created waves. If omitted, the command exports waves from the active design unit.

  <name> — Specifies a design unit in the simulation.

- -endtime {<time><unit>

  (required) The simulation time at which you want to stop exporting.

  <time> — Specified as an integer or decimal number.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- -file <filename>

  (required) The filename for the saved export file.

  <name> — Any user specified string.

- -format force | vcd | vhdl | verilog

  (required) The format of the saved stimulus file. The format options include:

  force — A Tcl script that recreates the waveforms. The file should be sourced when reloading the simulation.

  vcd — An extended VCD file. Load using the **-vcdstim** argument to **vsim**.

  vhdl — A VHDL test bench. Compile and load the file as your top-level design unit.

  verilog — A Verilog test bench. Compile and load the file as your top-level design unit.

- -starttime {<time><unit>}

  (required) The simulation time at which you want to start exporting.

  <time> — Specified as an integer or decimal number.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

Related Topics

- wave create
- wave import
- "Generating Stimulus with Waveform Editor"

wave import

Th**is** command imports an extended VCD file that was created with the **wave export** command. It cannot read extended VCD file created by software other than ModelSim. Use this command to apply a VCD file as stimulus to the current simulation.

Syntax

wave import **<VCD_file>**

Arguments

- <VCD_file>

  (required) The name of the extended VCD file to import.

Related Topics

- wave create
- wave export
- "Generating Stimulus with Waveform Editor"

wave modify

This command modifies waveform parameters set by a previous wave create command.

Some arguments to this command are order-dependent. Please read through the argument descriptions for more information.

The following table summarizes the available wave modification options:

Command	Description
wave modify -pattern clock	Generates a clock waveform. Specify an initial value, duty cycle, and clock period for the waveform.
wave modify -pattern constant	Generates a waveform with a constant value. Specify a value.
wave modify -pattern counter	Generates a waveform from a counting pattern. Specify start and end values, repeat, step count, time period, and type (Binary, Gray, Johnson, OneHot, Range, and ZeroHot).
wave modify -pattern random	Generates a random waveform based upon a seed value. Specify the type (normal or uniform), an initial value, and a seed value. If you don't specify a seed value, Questa uses a default value of 5.
wave modify -pattern repeater	Generates a waveform that repeats. Specify an initial value and pattern that repeats. You can also specify how many times the pattern repeats.
wave modify -pattern none	Creates a placeholder for a waveform. Specify an object name.

Syntax

All waveforms

wave modify [-driver freeze | deposit | driver | expectedoutput] [-endtime {<time><unit>}] [-initialvalue <value>] [-portmode {in | out | inout | internal}] [-range <msb lsb>] [-starttime {<time><unit>}] **<wave_name>**

Clock patterns only

wave modify -pattern clock **-period <value> -dutycycle <value> <wave_name>**

Constant patterns only

wave modify -pattern constant [-driver freeze | deposit | driver | expectedoutput] [-language {vhdl | verilog}] [**-value <value>**] **<wave_name>**

Counter patterns only

wave modify -pattern counter **-period <value> -repeat forever | <n> | never -startvalue <value> -step <value>** [-direction {up | down | upthendown | downthenup}]

[-endvalue <value>] [-type {binary | gray | johnson | onehot | range | zerohot}] **<wave_name>**

Random patterns only

wave modify -pattern random **-period <value>** **-random_type exponential | normal | poisson | uniform** [-seed <value>] **<wave_name>**

Repeater patterns only

wave modify -pattern repeater **-period <value>** **-repeat forever | <n> | never** **-sequence {val1 val2 val3 …}** **<wave_name>**

No pattern

wave create **-pattern none** **<wave_name>**

Arguments

- -direction {up | down | upthendown | downthenup}

  (optional, recommended when specifying **-pattern counter**) The direction in which the counter will increment or decrement.

  up — (default) Increment only.

  down — Decrement only.

  upthendown — Increment then decrement.

  downthenup — Decrement then increment.

- -driver freeze | deposit | driver | expectedoutput

  (optional) Specifies that the signal is a driver of the specified type. Applies to signals of type inout or internal.

- -dutycycle <value>

  (required) The duty cycle of the clock, expressed as a percentage of the period that the clock is high.

  <value> — Any integer from 0 to 100 where the default is 50.

- -endtime {<time><unit>}

  (optional) The simulation time that the waveform should stop. If omitted, the waveform stops at 1000 simulation time units.

  <time> — Specified as an integer or decimal number.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- -endvalue <value>

  (optional) The ending value of the counter. This option applies to Range counter patterns only. All other counter patterns start from 0 and go to the max value for that particular signal (for example, for a 3-bit signal, the start value will be 000 and end value will be 111).

  <value> — Any positive integer.

- -initialvalue <value>

  (optional) The initial value for the waveform. Value must be appropriate for the type of waveform you are creating. Not applicable to counter patterns.

  <value> — Any positive integer.

- -language {vhdl | verilog}

  (optional) Controls which language is used for modifying the wave.

  vhdl — (default) Specifies the VHDL language.

  verilog — Specifies the Verilog language.

- -period <value>

  (required) The period of the signal.

- -portmode {in | out | inout | internal}

  (optional) The port type for the waveform.

  in — Ports of type IN. You can also specify “input” as an alias for in.

  out — Ports of type OUT. You can also specify “output” as an alias for out.

  inout — Ports of type INOUT.

  internal — (default) Ports of type INTERNAL.

- -random_type exponential | normal | poisson | uniform

  (required) Specifies a random pattern to generate.

  exponential — Exponential distribution of waveform events.

  normal — Normal or Gaussian distribution of waveform events.

  poisson — Poisson distribution of waveform events.

  uniform — (default) Uniform distribution of waveform events.

- -range <msb lsb>

  (optional) Identifies bit significance in a counter pattern.

  msb lsb — Most significant bit and least significant bit. Both must be specified.

- -repeat forever | <n> | never

  (required) Controls duration of pattern repetition.

  forever — Repeat the pattern for as long as the simulation runs.

  <n> — Repeat the pattern <n> number of times where <n> is any positive integer.

never — Never repeat the pattern during simulation.

- -seed <value>

  (optional) Specifies a seed value for a randomly generated waveform.

  <value> — Any non-negative integer where the default is 5.

- -sequence {val1 val2 val3 …}

  (required) The set of values that you want repeated.

  <val1> — Value must be appropriate for the type of waveform you are creating. Multiple values are entered as a space separated list and enclosed in curly braces ({}).

- -starttime {<time><unit>}

  (optional) The simulation time that the waveform should start. If omitted, the waveform starts at 0 simulation time units.

  <time> — Specified as an integer or decimal number.

  <unit> — (optional) A suffix specifying a unit of time where the default is to specify the current simulation resolution by omitting **<unit>**. Valid time units are: fs, ps, ns, us, ms, sec, min, and hr. If **<unit>** is specified, you must enclose **<time>** and **<unit>** within curly braces ({}).

- -startvalue <value>

  (required when specifying **-pattern counter**) The initial value of the counter. This option applies to patterns specifying **-type Range** only. All other counter patterns start from 0 and go to the maximum value for that particular signal (e.g., for a 3-bit signal, the start value will be 000 and the end value will be 111).

  <value> — Value must be appropriate for the type of waveform you are creating.

- -step <value>

  (required) The step by which the counter is incremented/decremented.

  <value> — Value must be appropriate for the type of waveform you are creating.

- -type {binary | gray | johnson | onehot | range | zerohot}

  (optional) Specifies a counter format.

  binary — Specifies a binary counter.

  gray — Specifies a binary counter where two successive values differ in only one bit. Also known as a reflected binary counter.

  johnson — Specifies a twisted ring or Johnson counter.

  onehot — Specifies a shift counter where only one bit at a time is set to “on” (1).

  range — (default) Specifies a binary counter where the values range between **-startvalue** and **-endvalue**

  zerohot — Specifies a shift counter where only one bit at a time is set to “off” (0).

- -value <value>

  (optional, recommended when specifying **-pattern constant**) Specifies a value for the constant pattern.

  <value> — Value must be appropriate for the type of waveform you are creating.

- <wave_name>

  (required) The name of an existing waveform created with the wave create command.

Related Topics

- wave create
- "Generating Stimulus with Waveform Editor"
- "Creating Waveforms from Patterns".

wave sort

This command sorts signals in the Wave window by name or full path name.

Syntax

wave sort **{ascending | descending | fa | fd}**

Arguments

- **ascending | descending | fa | fd**

  (required) Sort signals in one of the following orders.

  ascending — Sort in ascending order by signal name.

  descending — Sort in descending order by signal name.

  fa — Sort in ascending order by the full path name.

  fd — Sort in descending order by full path name.

Examples

```
wave sort ascending
```

when

This command instructs ModelSim to perform actions when the specified conditions are met.

For example, you can use the command to break on a signal value or at a specific simulator time. Use the nowhen command to deactivate **when** commands.

Syntax

when [[-fast] [-id <id#>] [-label <label>] **{<when_condition_expression>} {<command>}**]

Description

The **when** command uses a **when_condition_expression** to determine whether or not to perform the action. Conditions can include VHDL signals and Verilog nets and registers. The **when_condition_expression** uses a simple restricted language (that is not related to Tcl), which permits only four operators and operands that may be either HDL object names, signame'event, or constants. ModelSim evaluates the condition every time any object in the condition changes, hence the restrictions.

Here are some additional points to keep in mind about the when command:

- The **when** command creates the equivalent of a VHDL process or a Verilog always block. It does not work like a looping construct you might find in other languages such as C.
- Virtual signals, functions, regions, types, and so forth, cannot be used in the **when** command. Neither can simulator state variables other than $now.
- With no arguments, **when** will list the currently active when statements and their labels (explicit or implicit).

Syntax

when [[-fast] [-id <id#>] [-label <label>] {<when_condition_expression>} {<command>}]

Embedded Commands Allowed with the -fast Argument

You can use any Tcl command as a <command>, along with any of the following **vsim** commands:

- bp, bd
- change
- disablebp, enablebp
- echo
- examine
- force, noforce
- log, nolog
- stop

- when, nowhen

Embedded Commands Not Allowed with the -fast Argument

- Any do commands
- Any Tk commands or widgets
- References to U/I state variables or tcl variables
- Virtual signals, functions, or types

Using Global Tcl Variables with the -fast Argument

Embedded commands that use global Tcl variables for passing a state between the when command and the user interface need to declare the state using the Tcl uivar command. For example, the variable i below is visible in the GUI. From the command prompt, you can display it (by entering echo $i) or modify it (for example, by entering set i 25).

```
set i 10
when -fast {clk == '0'} {
   uivar i
   set i [expr {$i - 1}]
   if {$i <= 0} {
      force reset 1 0, 0 250
   }
}
when -fast {reset == '0'} {
   uivar i
   set i 10
}
```

Additional Restrictions on the -fast Argument

Accessing channels (such as files, pipes, sockets) that were opened outside of the embedded command will not work. For example:

```
set fp [open mylog.txt w]
when -fast {bus} {
   puts $fp "bus change: [examine bus]"
}
```

The channel that $fp refers to is not available in the simulator, only in the user interface. Even using the uivar command does not work here because the value of $fp has no meaning in the context of the -fast argument.

The following method of rewriting this example opens the channel, writes to it, then closes it within the **when** command:

```
when -fast {bus} {
   set fp [open mylog.txt a]
   puts $fp "bus change: [examine bus]"
   close $fp
}
```

The following example is a little more sophisticated method of doing the same thing:

```
when -fast {$now == 0ns} {
   set fp [open mylog.txt w]
}
when -fast {bus} {
   puts $fp "bus change: [examine bus]"
}
when -fast {$now == 1000ns} {
   close $fp
}
```

The general principle is that any embedded command done using the -fast argument is global to all other commands used with the -fast argument. Here, {$now == 0ns} is a way to define Tcl processes that the -fast commands can use. These processes have the same restrictions that when bodies have, but the advantage is again speed as a proc will tend to execute faster than code in the when body itself.

It is recommended not to use virtual signals and expressions.

Arguments

- -fast

  (optional) Causes the embedded <command> to execute within the simulation kernel, which provides faster execution and reduces impact on simulation runtime performance. Limitations on using the -fast argument are described above (in "Embedded Commands Not Allowed with the -fast Argument"). Disallowed commands still work, but they slow down the simulation.

- -label <label>

  (optional) Used to identify individual **when** commands.

  <label> — Associates a name or label with the specified when command. Adds a level of identification to the when command. The label may contain special characters. Quotation marks (" ") or braces ({ }) are required only if **<label>** contains spaces or special characters.

- -id <id#>

  (optional) Attempts to assign this id number to the when command.

  <id#> — Any positive integer that is not already assigned. If the id number you specify is already used, ModelSim will return an error.

Note

Id numbers for when commands are assigned from the same pool as those used for the bp command. So even if you have not specified a given id number for a when command, that number may still be used for a breakpoint.

- {<when_condition_expression>}

  (required if a command is specified) Specifies the conditions to be met for the specified <command> to be executed. The condition is evaluated in the simulator kernel and can be an object name, in which case the curly braces can be omitted. The command will be

executed when the object changes value. The condition can be an expression with these operators:

Name	Operator
equals	==, =
not equal	!=, /=
greater than	>
less than	<
greater than or equal	>=
less than or equal	<=
AND	&&, AND
OR	||, OR

The operands may be object names, signame'event, or constants. Subexpressions in parentheses are permitted. The command will be executed when the expression is evaluated as TRUE or 1.

The formal BNF syntax is:

```
condition ::= Name | { expression }

expression ::= expression AND relation
               | expression OR relation
               | relation
relation ::= Name = Literal
           | Name /= Literal
           | Name ' EVENT
           | ( expression )
Literal ::= '<char>' | "<bitstring>" | <bitstring>
```

The "=" operator can occur only between a Name and a Literal. This means that you cannot compare the value of two signals, i.e., Name = Name is not possible.

Tcl variables can be used in the condition expression but you must replace the curly braces ({}) with double quotes (""). This works like a macro substitution where the Tcl variables are evaluated once and the result is then evaluated as the when condition. Condition expressions are evaluated in the **vsim** kernel, which knows nothing about Tcl variables. That's why the condition expression must be evaluated in the GUI before it is sent to the **vsim** kernel. See below for an example of using a Tcl variable.

The ">", "<", ">=", and "<=" operators are the standard ones for vector types, not the overloaded operators in the std_logic_1164 package. This may cause unexpected results when comparing objects that contain values other than 1 and 0. ModelSim does a lexical comparison (position number) for values other than 1 and 0. For example:

```
0000 < 1111 ## This evaluates to true
H000 < 1111 ## This evaluates to false
001X >= 0010 ## This also evaluates to false
```

- {<command>}

  (required if a when expression is specified) The command(s) for this argument are evaluated by the Tcl interpreter within the ModelSim GUI. Any ModelSim or Tcl command or series of commands are valid with one exception—the run command cannot be used with the **when** command. The command sequence usually contains a stop command that sets a flag to break the simulation run after the command sequence is completed. Multiple-line commands can be used.

Note

If you want to stop the simulation using a **when** command, you must use a **stop** command within your when statement. DO NOT use an exit command or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated.

Examples

- The **when** command below instructs the simulator to display the value of object *c* in binary format when there is a clock event, the clock is 1, and the value of *b* is 01100111. Finally, the command tells ModelSim to stop.

```
when -label when1 {clk'event and clk='1' and b = "01100111"} {
   echo "Signal c is [exa -bin c]"
   stop
}
```

- The **when** command below echoes the simulator time when slice [3:1] of wire [15:0] count matches the hexadecimal value 7, and simulation time is between 70 and 111 nanoseconds.

```
when {$now > 70ns and count(3:1) == 3'h7 && $now < 111ns} {
   echo "count(3:1) matched 3'h7 at time " $now
}
```

- The commands below show an example of using a Tcl variable within a **when** command. Note that the curly braces ({}) have been replaced with double quotes ("").

```
set clkb_path /tb/ps/dprb_0/udprb/ucar_reg/uint_ram/clkb;
when -label when1 "$clkb_path'event and  $clkb_path ='1'" {
     echo "Detected Clk edge at path $clkb_path"
}
```

- The **when** command below is labeled *a* and will cause ModelSim to echo the message "b changed" whenever the value of the object *b* changes.

```
when -label a b {echo "b changed"}
```

- The multi-line **when** command below does not use a label and has two conditions. When the conditions are met, ModelSim runs an echo command and a stop command.

```
when {b = 1
    and c /= 0 } {
    echo "b is 1 and c is not 0"
    stop
}
```

- In the example below, for the declaration "wire [15:0] a;", the **when** command will activate when the selected bits match a 7:

```
when {a(3:1) = 3'h7} {echo "matched at time " $now}
```

- In the example below, we want to sample the values of the address and data bus on the first falling edge of *clk* after *sstrb* has gone high.

```
# ::strobe is our state variable
set ::strobe Zero
# This signal breakpoint only fires when sstrb changes to a '1'
when -label checkStrobe {/top/sstrb == '1'} {
   # Our state Zero condition has been met, move to state One
   set ::strobe One
   }
# This signal breakpoint fires each time clk goes to '0'
when {/top/clk == '0'} {
   if {$::strobe eq "One"} {
      # Our state One condition has been met
      # Sample the busses
      echo Sample paddr=[examine -hex /top/paddr] :: sdata=[examine
-hex
      /top/sdata]
      # reset our state variable until next rising edge of sstrb
(back to
      state Zero)
      set ::strobe Zero
   }
}
```

Ending the simulation with the stop command

Batch mode simulations are often structured as "run until condition X is true," rather than "run for X time" simulations. The multi-line **when** command (shown below) sets a done condition, and ModelSim runs an echo command and a stop command when the condition is reached.

The simulation will not stop (even if a **quit -f** command is used) unless you enter a **stop** command. To exit the simulation and quit ModelSim, use an approach like the following:

```
onbreak {resume}
when {/done_condition == '1'} {
     echo "End condition reached"
     if [batch_mode] {
          set DoneConditionReached 1
          stop
     }
}
run 1000 us
if {$DoneConditionReached == 1} {
     quit -f
}
```

This example stops 100ns after a signal transition:

```
when {a = 1} {
  # If the 100ns delay is already set then let it go.
  if {[when -label a_100] == ""} {
    when -label a_100 { $now = 100 } {
      # delete this breakpoint then stop
      nowhen a_100
      stop
    }
  }
}
```

Time-based breakpoints

You can build time-based breakpoints into a **when** statement with the following syntax.

For absolute time (indicated by @) use:

```
when {$now = @1750 ns} {stop}
```

You can also use:

```
when {errorFlag = '1' OR $now = 2 ms} {stop}
```

This example adds 2 ms to the simulation time at which the **when** statement is first evaluated, then stops. The white space between the value and time unit is required for the time unit to be understood by the simulator.

You can also use variables, as shown in the following example:

```
set time 1000
when "\$now = $time" {stop}
```

The quotes instruct Tcl to expand the variables before calling the command. So, the **when** command sees:

```
when "$now = 1000" stop
```

Note that "$now" has the '$' escaped. This prevents Tcl from expanding the variable, because if it did, you would get:

```
when "0 = 1000" stop
```

Related Topics

- bp
- disablebp
- enablebp
- nowhen

where

This command displays information about the system environment. It is useful for debugging problems where ModelSim cannot find the required libraries or support files.

The command displays two results on consecutive lines:

- current directory

  This is the current directory that ModelSim was invoked from, or that was specified on the ModelSim command line.

- current project file

  This is the *.mpf* file ModelSim is using. All library mappings are taken from here when a project is open. If the design is not loaded through a project, this line displays the *modelsim.ini* file in the current directory.

Syntax

where

Arguments

- None.

Examples

- Design is loaded through a project:

  VSIM> where

  Returns:

  ```
  #  Current directory is: D:\Client
  #  Project is: D:/Client/monproj.mpf
  ```

- Design is loaded with no project (indicates the modelsim.ini file is under the `mydesign` directory):

  VSIM> where

  Returns:

  ```
  #  Current directory is: C:\Client\testcase\mydesign
  #  Project is: modelsim.ini
  ```

wlf2log

This command translates a ModelSim WLF file (*vsim.wlf*) to a QuickSim II logfile. It reads the *vsim.wlf* WLF file generated by the add list, add wave, or log commands in the simulator and converts it to the QuickSim II logfile format.

Note

This command should be invoked only after you have stopped the simulation using **quit -sim** or **dataset close sim**.

Syntax

wlf2log **<wlffile>** [-bits] [-fullname] [-help] [-inout] [-input] [-internal] [-l <instance_path>] [-lower] [-o <outfile>] [-output] [-quiet]

Arguments

- <wlffile>

  (required) Specifies the ModelSim WLF file that you are converting.

- -bits

  (optional) Forces vector nets to be split into 1-bit wide nets in the log file.

- -fullname

  (optional) Shows the full hierarchical pathname when displaying signal names.

- -help

  (optional) Displays a list of command options with a brief description for each.

- -inout

  (optional) Lists only the inout ports. This may be combined with the -input, -output, or -internal switches.

- -input

  (optional) Lists only the input ports. This may be combined with the -output, -inout, or -internal switches.

- -internal

  (optional) Lists only the internal signals. This may be combined with the -input, -output, or -inout switches.

- -l <instance_path>

  (optional) Lists the signals at or below an HDL instance path within the design hierarchy.

  <instance_path> — Specifies an HDL instance path.

- -lower

  (optional) Shows all logged signals in the hierarchy. When invoked without the -lower switch, only the top-level signals are displayed.

- -o <outfile>

  (optional) Directs the output to be written to a file where the default destination for the logfile is standard out.

  <outfile> — A user specified filename.

- -output

  (optional) Lists only the output ports. This may be combined with the -input, -inout, or -internal switches.

- -quiet

  (optional) Disables error message reporting.

wlf2vcd

This command translates a ModelSim WLF file to a standard VCD file. Complex data types that are unsupported in the VCD standard (records, memories, etc.) are not converted.

Note

This command should be invoked only after you have stopped the simulation using **quit -sim** or **dataset close sim**.

Syntax

wlf2vcd **<wlffile>** [-help] [-o <outfile>] [-quiet]

Arguments

- <wlffile>

  (required) Specifies the ModelSim WLF file that you are converting.

- -help

  (optional) Displays a list of command options with a brief description for each.

- -o <outfile>

  (optional) Specifies a filename for the output where the default destination for the VCD output is stdout.

  <outfile> — A user specified filename.

- -quiet

  (optional) Disables warning messages that are produced when an unsupported type (for example, records) is encountered in the WLF file.

wlfman

This command allows you to get information about and manipulate saved WLF files.

The command performs four functions depending on which mode you use:

- **wlfman info** returns file information, resolution, versions, and so forth about the source WLF file.
- **wlfman items** generates a list of HDL objects (i.e., signals) from the source WLF file and outputs it to stdout. When redirected to a file, the output is called an object_list_file, and it can be read in by **wlfman filter**. The object_list_file is a list of objects, one per line. Comments start with a '#' and continue to the end of the line. Wildcards are legal in the leaf portion of the name. Here is an example:

  ```
  /top/foo     # signal foo
  /top/u1/*    # all signals under u1
  /top/u1      # same as line above
  -r /top/u2   # recursively, all signals under u2
  ```

  Note that you can produce these files from scratch but be careful with syntax. **wlfman items** always creates a legal object_list_file.
- **wlfman filter** reads in a WLF file and, optionally, an object_list_file, and writes a new WLF file containing filtered information from those sources. You determine the filtered information with the arguments you specify.
- **wlfman monitor** returns the current state of a WLF file to the transcript. Each time the state is monitored, a line of information is output. The state of the WLF file can be monitored at regular intervals, indicating the changes over time. Here is an example:

  ```
  wlfman monitor visim.wlf
  File    Sim
  State   Time
  closed  14000
  ```
- **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 (such as a logged memory that uses a zero-delay integer loop to initialize the memory). You may be able to drastically reduce WLF file size by not logging those signals.
- **wlfman merge** combines two WLF files with different signals into one WLF file. It *does not* combine wlf files containing the same signals at different runtime ranges (i.e., mixedhdl_0ns_100ns.wlf & mixedhdl_100ns_200ns.wlf).
- **wlfman optimize** copies the data from the WLF file to the output WLF file, adding or replacing the indexing and optimization information.

The different modes are intended to be used together. For example, you might run **wlfman profile** and identify a signal that accounts for 50% of the WLF file size. If you don’t actually need that signal, you can then run **wlfman filter** to remove it from the WLF file.

Syntax

wlfman info **<source_wlffile>** [-v]

wlfman items **<source_wlffile>** [-n] [-v]

wlfman filter **-o <out_wlffile> <source_wlffile>** [-begin <time>] [-end <time>] [-compress | -nocompress] [-f <object_list_file>] [-index | -noindex] [-r <object>] [-opt | -noopt] [-s <symbol>] [-t <resolution>]

wlfman profile **<source_wlffile>** [-rank] [-top <number>]

wlfman merge **-o <out_wlffile>** [**<wlffile1> <wlffile2> ...**] [-compress | -nocompress] [-index | -noindex] [-opt | -noopt]

wlfman monitor [-f | -i <intervalTime> | -p <endTime>] [-q | -v] **<source_wlffile>**

wlfman optimize **-o <out_wlffile> <source_wlffile>** [-opt | -noopt]

Arguments

- -o <out_wlffile>

  (required) Specifies the name of the output WLF file. The output WLF file will contain all objects specified by **the preceding arguments**. Output WLF files are always written in the latest WLF version regardless of the source WLF file version.

- <source_wlffile>

  (required) Specifies the WLF file from which you want information.

- <wlffile1> <wlffile2> ...

  (required) Specifies the WLF files whose objects you want to copy into one WLF file. Specified as a space separated list.

- -begin <time>

  (optional) Specifies the simulation time at which to start reading information from the source WLF file where the default is to include the entire length of time recorded in **<source_wlffile>**.

- -compress | -nocompress

  (optional) Controls compression of the output WLF file.

  -compress — Enables compression. (default)

  -nocompress — Disables compression.

- -end <time>

  (optional) Specifies the simulation time at which you want to end reading information from **<source_wlffile>**.

- -f

  (optional) Repeat status update every 10 seconds of real time unless an alternate time interval is specified with -i <intervalTime>.

- -f <object_list_file>

  (optional) Specifies an object_list_file created by **wlfman items** to include in **<out_wlffile>**.

- -i <intervalTime>

  (optional) Specifies the time delay before the next status update where the default is 10 seconds of real time if not specified.

  <intervalTime> — Any positive integer.

- -index | -noindex

  (optional) Controls indexing when writing the output WLF file. Indexing makes viewing wave data faster, however performance during optimization will be slower because indexing and optimization require significant memory and CPU resources. Disabling indexing makes viewing wave data slower unless the display is near the start of the WLF file. Disabling indexing also disables optimization of the WLF file but may provide a significant performance boost when archiving WLF files. Indexing and optimization information can be added back to the file using the **wlfman optimize command**.

  -index — Enables indexing. (default)

  -noindex — Disables indexing and optimization.

- -n

  (optional) Lists regions only (no signals).

- -opt | -noopt

  (optional) Controls optimization of **the output WLF file**.

  -opt — Enables WLF file optimization. (default)

  -noopt — Disables WLF file optimization.

- -p <endTime>

  (optional) Specifies the simulation time at which wlfman will stop monitoring the WLF file.

  <endTime> — Any positive integer.

- -q

  (optional) Suppress normal status messages while monitoring.

- -r <object>

  (optional) Specifies an object (region) to recursively include in the output. If <object> is a signal, the output would be the same as using **-s**.

- -rank

  (optional) Sorts the **wlfman profile** report by percentage of the total file space used by each signal.

- -s <symbol>

  (optional) Specifies an object to include in the output. By default all objects are included.

- -t <resolution>

  (optional) Specifies the time resolution of the new WLF file. By default the resolution is the same as the source WLF file.

- -top <number>

  (optional) Filters the **wlfman profile** report so that only the top <number> signals in terms of file space percentage are displayed.

- -v

  (optional) Produces verbose output that lists the object type next to each object.

Examples

- The output from this command would look something like this:

  wlfman profile -rank top_vh.wlf

  Returns:

```
#Repeated ID #'s mean those signals share the same
#space in the wlf file.
#
# ID     Transitions   File %   Name
#-----   -----------   ------   ------------------------------------
    1          2192      33 %    /top_vh/pdata
    1                            /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
```

- wlfman monitor -f -p 100000000 vsim.wlf

Returns:

```
Setting end time to 100000000, measuring progress %
File    File     Percent
State   Time     Complete
open    7239185  7.2%
open    7691785  7.7%
open    8144385  8.1%
open    8596625  8.6%
```

Related Topics

- Recording Simulation Results With Datasets
- WLF File Parameter Overview

wlfrecover

This command attempts to "repair" WLF files that are incomplete due to a crash or if the file was copied prior to completion of the simulation. Use this command if you receive a “bad magic number” error message when opening a WLF file. You can run the command from the VSIM> or ModelSim> prompt or from a shell.

Syntax

wlfrecover **<filename>** [-force] [-q]

Arguments

- <filename>

  (required) Specifies the WLF file to repair.

- -force

  (optional) Disregards file locking and attempts to repair the file.

- -q

  (optional) Hides all messages unless there is an error while repairing the file.

Related Topics

- Saving a Simulation to a WLF File

write format

This command records the names and display options of the HDL objects currently being displayed in the Analysis, List, Memory, Message Viewer, Test Browser, and Wave windows.

The **write format restart** command creates a single *.do* file that will recreate all debug windows, all file/line breakpoints, and all signal breakpoints created using the when command. If the ShutdownFile *modelsim.ini* variable is set to this *.do* filename, it will call the write format restart command upon exit.

The file created is primarily a list of add list or add wave commands, though a few other commands are included (see "Output" below). This file may be invoked with the do command to recreate the window format on a subsequent simulation run.

When you load a format file, ModelSim verifies the existence of the datasets required by that file. ModelSim displays an error message if the requisite datasets do not all exist. To force the execution of the format file even if all datasets are not present, use the **-force** switch with your **do** command. For example:

```
VSIM> do format.do -force
```

Note

Note that using the -force switch when datasets are not present will result in error messages for signals referencing the nonexistent datasets. Also, -force is recognized by the format file not the **do** command.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

write format **{assertions | breakpoints | coverdirective | export_hier_config | list | memory | msgviewer | testbrowser | watch | wave | restart}**
<filename>

Arguments

- assertions | breakpoints | coverdirective | export_hier_config | list | memory | msgviewer | testbrowser | watch | wave | restart

  (required) Specifies that the contents of the designated window are recorded in the file specified by <filename>.

  assertions — Records objects of the Assertions window.

  breakpoints — Records file line and signal breakpoints.

  coverdirective — Records objects of the Coverdirectives window.

  export_hier_config — Records hierarchical sort order for objects in the Verification Results Analysis window.

  list — Records objects of the List window.

memory — Records objects of the Memory window.

msgviewer — Records objects of the Message Viewer window.

testbrowser — Records objects of the Verification Management Browser window.

watch — Records objects of the Watch window.

wave — Records objects of the Wave window.

restart — Records objects of all windows and breakpoints in the *.do* file.

- <filename>

  (required) Specifies the name of the output file where the data is to be written. You must specify the *.do* extension.

Examples

- Save the current data in the List window in a file named *alu_list.do*.

```
write format list alu_list.do
```

- Save the current data in the Wave window in a file named *alu_wave.do*.

```
write format wave alu_wave.do
```

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. 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.

Related Topics

- add list
- add wave
-

write list

This command records the contents of the List window in a list output file.

This file contains simulation data for all HDL objects displayed in the List window: VHDL signals and variables and Verilog nets and registers.

Arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

write list [-events] <**filename**>

Arguments

- -events

  (optional) Specifies to write print-on-change format where the default is tabular format.

- <filename>

  (required) Specifies the name of the output file where the data is to be written.

Examples

- Save the current data in the List window in a file named *alu.lst*.

```
write list alu.lst
```

Related Topics

- write tssi

write preferences

This command saves the current GUI preference settings to a Tcl preference file. Settings saved include Wave, Objects, and Locals window column widths; Wave, Objects, and Locals window value justification; and Wave window signal name width.

Syntax

write preferences **<preference file name>**

Arguments

- <preference file name>

  (required) Specifies the name for the preference file. If the file is named *modelsim.tcl*, ModelSim will read the file each time **vsim** is invoked. To use a preference file other than *modelsim.tcl* you must specify the alternative file name with the MODELSIM_TCL environment variable.

You can modify variables by editing the preference file with the ModelSim notepad:

```
notepad <preference file name>
```

write report

This command prints a summary of the design being simulated including a list of all design units (VHDL configurations, entities, and packages, and Verilog modules) with the names of their source files. The summary includes a list of all source files used to compile the given design.

The Simulation Report contains the following information:

- Design Simulated — directory path of the design's top-level module
- Number of signals/nets in the design
- Number of processes in the design
- Simulator Parameters, including:
  - Current directory
  - Project file directory
  - Simulation time resolution
- List of design units used, including:
  - Module name
  - Architecture, if applicable
  - Library directory
  - Source file
  - Timescale
  - Occurrences

Some arguments to this command are order-dependent. Please read through the argument descriptions for more information.

Syntax

write report [-capacity [-l | -s] [-qdas]] | [-l | -s] |
[-tcl] | [<filename>]

Arguments

- -capacity

  (optional) Reports data on memory usage of various types of SystemVerilog constructs in the design. ModelSim collects memory usage data for dynamic objects. Must be specified first when specifying -qdas. To display memory data for all object types, specify -capacity -l.

- <filename>

  (optional) Specifies the name of the output file where the data is to be written. If <filename> is omitted, the report is written to the Transcript window.

- -l

  (optional) Generates more detailed information about the design, including a list of sparse memories or the memory capacity for all object types. You must precede this argument with -capacity when specifying a capacity report.

- -qdas

  (optional) Reports memory usage data for queues, dynamic arrays, and associative arrays. You must precede this argument with -capacity when specifying a capacity report.

- -s

  (optional) Generates a short list of design information. You must precede this argument with -capacity when specifying a capacity report.

- -tcl

  (optional) Generates a Tcl list of design unit information. This argument cannot be used with a filename.

Examples

- Save information about the current design in a file named *alu_rpt.txt*.

  write report alu_rpt.txt

- Create a Simulation Report for the current simulation

  write report -l

  returns:

```
##
## SIMULATION REPORT          Generated on Mon Aug 10 12:56:15 2009
##
##
## Design simulated: <directory>\work.top(fast)
## Number of signals/nets in design: 89
## Number of processes in design: 74
##
## Simulator Parameters:
##
##     Current directory: <directory>\
##     Project file: <directory>\win32/../modelsim.ini
##     Simulation time resolution: 1ns
##
## List of Design units used:
##
##     Module: top
##     Architecture: fast
##     Library: <directory>\work
```

```
##     Source File: top.v
##     Timescale: 1ns / 1ns
##     Occurrences: 1
##
##     Module: proc
##     Architecture: fast
##     Library: <directory>\work
##     Source File: proc.v
##     Timescale: 1ns / 1ns
##     Occurrences: 1

...
```

write timing

This command displays path delays and timing check limits, unadjusted for delay net delays, for the specified instance.

When the **write timing** command reports interconnect delays on a Verilog module instance you will see either MIPDs (Module Input Port Delays) or MITDs (Module Transport Port Delays) reported. If you specify either the **+multisource_int_delays** or the **+transport_int_delays** switch with the vsim command, INTERCONNECT delays will be reported as MITDs. Otherwise they will be reported as MIPDs. An MIPD report may look like the following:

```
# /top/u1: [mymod:src/5/test.v(18)]
#   MIPD(s):
#     Port clk_in: (6, 6, 6)
```

An MITD report may look like the following:

```
# /top/u1: [mymod:src/5/test.v(18)]
#   MITDs to port clk_in:
#     From port /top/p/y = (6)
```

Syntax

write timing [-recursive] [-file <filename>] [<instance_name1>…<instance_nameN>] [-simvalues]

Arguments

- -file <filename>

  (optional) Specifies the name of the output file where the data is to be written. If the -file argument is omitted, timing information is written to the Transcript window.

  <filename> — Any valid filename. May include special characters and numbers.

- <instance_name1>…<instance_nameN>

  (required) The name(s) of the instance(s) for which timing information will be written. If <instance_name> is omitted, the command returns nothing.

- -recursive

  (optional) Generates timing information for the specified instance and all instances underneath it in the design hierarchy.

- -simvalues

  (optional) Displays optimization-adjusted values for delay net delays.

Examples

- Write timing about */top/u1* and all instances underneath it in the hierarchy to the file *timing.txt*.

  write timing -r -f timing.txt /top/u1

- Write timing information about the designated instances to the Transcript window.

  write timing /top/u1 /top/u2 /top/u3 /top/u8

write transcript

This command writes the contents of the Transcript window to the specified file. The resulting file can then be modified to replay the transcribed commands as a DO file (macro).

The command cannot be used in batch mode. In batch mode use the standard Transcript file or redirect stdout.

Syntax

write transcript [<filename>]

Arguments

- <filename>

  (optional) Specifies the name of the output file where the data is to be written. If the <filename> is omitted, the transcript is written to a file named *transcript*.

Related Topics

- do
- Saving a Transcript File as a Macro (DO file)

write tssi

This command records the contents of the List window in a "TSSI format" file.

The file contains simulation data for all HDL objects displayed in the List window that can be converted to TSSI format (VHDL signals and Verilog nets). A signal definition file is also generated.

The List window needs to be using symbolic radix in order for **write tssi** to produce useful output.

Syntax

write tssi **<filename>**

Arguments

- <filename>

  (required) Specifies the name of the output file where the data is to be written.

Description

If the <filename> has a file extension (e.g., *listfile.lst*), then the definition file is given the same file name with the extension *.def* (e.g., *listfile.def*). The values in the listfile are produced in the same order that they appear in the List window. The directionality is determined from the port type if the object is a port, otherwise it is assumed to be bidirectional (mode INOUT).

Objects that can be converted to SEF are VHDL enumerations with 255 or fewer elements and Verilog nets. The enumeration values U, X, 0, 1, Z, W, L, H and - (the enumeration values defined in the IEEE Standard 1164 **std_ulogic** enumeration) are converted to SEF values according to the table below. Other values are converted to a question mark (?) and cause an error message. Though the **write tssi** command was developed for use with **std_ulogic**, any signal which uses only the values defined for **std_ulogic** (including the VHDL standard type **bit**) will be converted.

std_ulogic State Characters	**SEF State Characters**		
	Input	**Output**	**Bidirectional**
U	N	X	?
X	N	X	?
0	D	L	0
1	U	H	1
Z	Z	T	F
W	N	X	?
L	D	L	0
H	U	H	1

std_ulogic State Characters	SEF State Characters		
	Input	**Output**	**Bidirectional**
-	N	X	?

Bidirectional logic values are not converted because only the resolved value is available. The TSSI TDS ASCII In Converter and ASCII Out Converter can be used to resolve the directionality of the signal and to determine the proper forcing or expected value on the port. Lowercase values x, z, w, l, and h are converted to the same values as the corresponding capitalized values. Any other values will cause an error message to be generated the first time an invalid value is detected on a signal, and the value will be converted to a question mark (?).

Note

The TDS ASCII In Converter and ASCII Out Converter are part of the TDS software. ModelSim outputs a vector file, and TSSI tools determine whether the bidirectional signals are driving or not.

Related Topics

- tssi2mti

write wave

This command records the contents of the Wave window in PostScript format.

The output file can then be printed on a PostScript printer.

Syntax

write wave **<filename>** [-end <time>] [-landscape] [-height <real_num>]
[-margin <real_num>] [-perpage <time>] [-portrait][-start <time>] [-width <real_num>]

Arguments

- <filename>

  (required) Specifies the name of the PostScript *(.ps)* output file.

- -end <time>

  (optional) The simulation time at which the record will end.

  <time> — Specified as a positive integer or decimal number where the units are the current simulation time resolution.

- -height <real_num>

  (optional) Specifies the paper height in inches.

  <real_num> — Specified as a positive integer or decimal number where the default is 11.0.

- -landscape

  (optional) Use landscape (horizontal) orientation. (default)

- -margin <real_num>

  (optional) Specifies the margin in inches.

  <real_num> — Specified as a positive integer or decimal number where the default is 0.5.

- -perpage <time>

  (optional) Specifies the time width per page of output.

  <time> — Specified as a positive integer or decimal number where the units are the current simulation time resolution.

- -portrait

  (optional) Use portrait (vertical) orientation where the default is landscape (horizontal).

- -start <time>

  (optional) Specifies the start time to be written.

  <time> — Specified as a positive integer or decimal number where the units are the current simulation time resolution.

- -width <real_num>

  (optional) Specifies the paper width in inches.

  <real_num> — Specified as a positive integer or decimal number where the default is 8.5.

Examples

- Save the current data in the Wave window in a file named *alu.ps*.

  ```
  write wave alu.ps
  ```

- Write two separate pages to *top.ps*. The first page contains data from 600ns to 700ns, and the second page contains data from 701ns to 800ns.

  ```
  write wave -start 600ns -end 800ns -perpage 100ns top.ps
  ```

  To make the job of creating a PostScript waveform output file easier, use the **File > Print Postscript** menu selection in the Wave window.

Index

— F —

— G —

— H —

— I —

— K —

— L —

— M —

— Q —

— R —

— S —

— T —

— U —

— V —

— W —

— X —

— Y —

— Z —

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **ORDERS, FEES AND PAYMENT.**

   1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement ("Order(s)"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not these documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order will not be effective unless agreed in writing by an authorized representative of Customer and Mentor Graphics.

   1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice(s). Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.

   1.3. All Products are delivered FCA factory (Incoterms 2000), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer requests any change or enhancement to Software, whether in the course of receiving support or consulting services, evaluating Software, performing beta testing or otherwise, any inventions, product

improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

4.1. Portions or all of certain Software may contain code for experimental testing and evaluation ("Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. This grant and Customer's use of the Beta Code shall not be construed as marketing or offering to sell a license to the Beta Code, which Mentor Graphics may choose not to release commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use it except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer learns or becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Software or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Software, or disclose to any third party the results of, or information pertaining to, any benchmark.

5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.

5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. 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/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at http://supportnet.mentor.com/about/legal/.

7. **AUTOMATIC CHECK FOR UPDATES; PRIVACY.** Technological measures in Software may communicate with servers of Mentor Graphics or its contractors for the purpose of checking for and notifying the user of updates and to ensure that the Software in use is licensed in compliance with this Agreement. Mentor Graphics will not collect any personally identifiable data in this process and will not disclose any data collected to any third party without the prior written consent of Customer, except to Mentor Graphics' outside attorneys or as may be required by a court of competent jurisdiction.

8. **LIMITED WARRANTY.**

8.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification or improper installation. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY, PROVIDED CUSTOMER HAS OTHERWISE COMPLIED WITH THIS AGREEMENT. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

8.2. THE WARRANTIES SET FORTH IN THIS SECTION 8 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

9. **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 RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). NEITHER MENTOR GRAPHICS NOR ITS LICENSORS SHALL BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF PRODUCTS AS DESCRIBED IN SECTION 10. THE PROVISIONS OF THIS SECTION 11 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

12. **INFRINGEMENT.**

12.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to the action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

12.2. If a claim is made under Subsection 12.1 Mentor Graphics may, at its option and expense, (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

12.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

12.4. THIS SECTION 12 IS SUBJECT TO SECTION 9 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS FOR DEFENSE, SETTLEMENT AND DAMAGES, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

13. **TERMINATION AND EFFECT OF TERMINATION.** If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term.

13.1. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

13.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

14. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States government agencies, which prohibit export or diversion of certain products and information about the products to certain countries and certain persons. Customer agrees that it will not export Products in any manner without first obtaining all necessary approval from appropriate local and United States government agencies.

15. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. All Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to US FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. Government or a U.S. Government subcontractor is subject solely to the terms and conditions set forth in this Agreement, except for provisions which are contrary to applicable mandatory federal laws.

16. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

17. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FLEXlm or FLEXnet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 17 shall survive the termination of this Agreement.

18. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the United States. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, USA, if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. This section shall not

restrict Mentor Graphics' right to bring an action against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

19. **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.

20. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements, including but not limited to any purchase order terms and conditions. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing by authorized representatives of the parties. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 100615, Part No. 246066