ModelSim EE/SE

Command Reference

Version 5.3

The ModelSim Elite and Special Editions

for VHDL, Verilog, and Mixed-HDL Simulation

Software Version: 5.3a

Published: September 1999

Model Technology Incorporated
10450 SW Nimbus Avenue / Bldg. R-B
Portland OR 97223-4347 USA

phone: 503-641-1340
fax: 503-526-5410
e-mail: support@model.com, sales@model.com
home page: http://www.model.com

EE Command Reference - Part # M16545 US$50

Software License Agreement

This is a legal agreement between you, the end user, and Model Technology Incorporated (MTI). By opening the sealed package you are agreeing to be bound by the terms of this agreement. If you do not agree to the terms of this agreement, promptly return the unopened package and all accompanying items to the place you obtained them for a full refund.

Model Technology Software License

1. LICENSE. MTI grants to you the **nontransferable**, **nonexclusive** right to use one copy of the enclosed software program (the "SOFTWARE") for each license you have purchased. The SOFTWARE must be used on the computer hardware server equipment that you identified in writing by make, model, and workstation or host identification number and the equipment served, in machine-readable form only, as allowed by the authorization code provided to you by MTI or its agents. All authorized systems must be used within the country for which the systems were sold. Model*Sim* licenses must be located at a single site, i.e. within a one-kilometer radius identified in writing to MTI. This restriction does not apply to single ModelSim PE licenses locked by a hardware security key, and such Model*Sim* PE products may be relocated within the country for which sold.

2. COPYRIGHT. The SOFTWARE is owned by MTI (or its licensors) and is protected by United States copyright laws and international treaty provisions. Therefore you must treat the SOFTWARE like any other copyrighted material, except that you may either (a) make one copy of the SOFTWARE solely for backup or archival purposes, or (b) transfer the SOFTWARE to a single hard disk provided you keep the original solely for backup or archival purposes. You may not copy the written materials accompanying the SOFTWARE.

3. USE OF SOFTWARE. The SOFTWARE is licensed to you for internal use only. You shall not conduct benchmarks or other evaluations of the SOFTWARE without the advance written consent of an authorized representative of MTI. You shall not sub-license, assign or otherwise transfer the license granted or the rights under it without the prior written consent of MTI or its applicable licensor. You shall keep the SOFTWARE in a restricted and secured area and shall grant access only to authorized persons. You shall not make software available in any form to any person other than your employees whose job performance requires access and who are specified in writing to MTI. MTI may enter your business premises during normal business hours to inspect the SOFTWARE, subject to your normal security.

4. PERMISSION TO COPY LICENSED SOFTWARE. You may copy the SOFTWARE only as reasonably necessary to support an authorized use. Except as permitted by Section 2, youmay not make copies, in whole or in part, of the SOFTWARE or other material provided by MTI without the prior written consent of MTI. For such permitted copies, you will include all notices and legends embedded in the SOFTWARE and affixed to its medium and container as received

from MTI. All copies of the SOFTWARE, whether provided by MTI or made by you, shall remain the property of MTI or its licensors.

You will maintain a record of the number and location of all copies of the SOFTWARE made, including copes that have been merged with other software, and will make those records available to MTI or its applicable licensor upon request.

5. TRADE SECRET. The source code of the SOFTWARE is trade secret or confidential information of MTI or its licensors. You shall take appropriate action to protect the confidentiality of the SOFTWARE and to ensure that any user permitted access to the SOFTWARE does not provide it to others. You shall take appropriate action to protect the confidentiality of the source code of the SOFTWARE. You shall not reverse-assemble, reverse-compile or otherwise reverse-engineer the SOFTWARE in whole or in part. The provisions of this section shall survive the termination of this Agreement.

6. TITLE. Title to the SOFTWARE licensed to you or copies thereof are retained by MTI or third parties from whom MTI has obtained a licensing right.

7. OTHER RESTRICTIONS. You may not rent or lease the SOFTWARE. You shall not mortgage, pledge or encumber the SOFTWARE in any way. You shall ensure that all support service is performed by MTI or its designated agents. You shall notify MTI of any loss of the SOFTWARE.

8. TERMINATION. MTI may terminate this Agreement, or any license granted under it, in the event of breach or default by you. In the event of such termination, all applicable SOFTWARE shall be returned to MTI or destroyed.

9. EXPORT. You agree not to allow the MTI SOFTWARE to be sent or used in any other country except in compliance with this license and applicable U.S. laws and regulations. If you need advice on export laws and regulations, you should contact the U.S. Department of Commerce, Export Division, Washington, DC 20230, USA for clarification.

Important Notice

Any provision of Model Technology Incorporated SOFTWARE to the U.S. Government is with "Restricted Rights" as follows: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause at FAR 2.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clauses in the NASA FAR Supplement. Any provision of Model Technology documentation to the U.S. Government is with Limited Rights. Contractor/manufacturer is Model Technology Incorporated, 10450 SW Nimbus Avenue / Bldg. R-B, Portland, Oregon 97223 USA.

Limited Warranty

LIMITED WARRANTY. MTI warrants that the SOFTWARE will perform substantially in accordance with the accompanying written materials for a period of 30 days from the date of receipt. Any implied warranties on the SOFTWARE are limited to 30 days. Some states do not allow limitations on duration of an implied warranty, so the above limitation may not apply to you.

CUSTOMER REMEDIES. MTI's entire liability and your exclusive remedy shall be, at MTI's option, either (a) return of the price paid or (b) repair or replacement of the SOFTWARE that does not meet MTI's Limited Warranty and which is returned to MTI. This Limited Warranty is void if failure of the SOFTWARE has resulted from accident, abuse or misapplication. Any replacement SOFTWARE will be warranted for the remainder of the original warranty period or 30 days, whichever is longer.

NO OTHER WARRANTIES. MTI disclaims all other warranties, either express or implied, including but not limited to implied warranties of merchantability and fitness for a particular purpose, with respect to the SOFTWARE and the accompanying written materials. This limited warranty gives you specific legal rights. You may have others, which vary from state to state.

NO LIABILITY FOR CONSEQUENTIAL DAMAGES. In no event shall MTI or its suppliers be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or other pecuniary loss) arising out of the use of or inability to use these MTI products, even if MTI has been advised of the possibility of such damages. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitation may not apply to you.

Table of Contents

ModelSim Commands (CR-9)

Model*Sim* Commands

Chapter contents

The simulator commands used to control the Model*Sim* simulator are described in this chapter. These commands are only valid after loading a design with the **vsim** command (CR-208) or via the Model*Sim* graphical interface. The commands here are entered either in macro files or on the command line of the Main window. Some commands are automatically entered on the command line when you use the Model*Sim* graphical user interface.

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

Command syntax

See "Command Syntax and Conventions" (CR-241) for complete command syntax information.

Note: All commands entered within Model*Sim* must be entered in lower case.

Tcl and FLEXlm commands

Documentation for Tcl commands, and FlexLM commands are available in these locations:

- **Tcl man pages**
  available in HTML format - use this Main window menu selection:
  Help > Tcl man pages
- **FLEXlm commands**
  for tools available within Model*Sim* see "License administration tools" (D-434) in the User's Manual; the complete FLEXlm user's manuals is available at:
  http://www.globetrotter.com/manual.htm

Command reference table

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

Command name	Action
.main clear (CR-99)	clears the Main window (10-158) transcript
.wave.tree zoomfull (CR-221)	redraws the display to show the entire simulation from time 0 to the current simulation time
.wave.tree zoomrange (CR-222)	enter the beginning and ending times for a range of time units to be displayed
abort (CR-19)	halts the execution of a macro file interrupted by a breakpoint or error
add button (CR-20)	adds a user-defined button to the Main window button bar
add list (CR-22)	lists VHDL variables, and Verilog nets and registers, and their values in the List window
add log	also known as the **log** command. See **log** (CR-93)
add wave (CR-33)	adds VHDL signals, and Verilog nets and registers to the Wave window
add_menu (CR-26)	adds a menu to the menu bar of the specified window, using the specified menu name
add_menucb (CR-28)	creates a checkbox within the specified menu of the specified window
add_menuitem (CR-30)	creates a menu item within the specified menu of the specified window
add_separator (CR-31)	adds a separator as the next item in the specified menu path in the specified window
add_submenu (CR-32)	creates a cascading submenu within the specified menu_path of the specified window
alias (CR-37)	creates a new Tcl procedure that evaluates the specified commands

Command name	Action
batch_mode (CR-38)	returns a 1 if VSIM is operating in batch mode, otherwise returns a 0
bd (CR-39)	deletes a breakpoint
bp (CR-40)	sets a breakpoint
cd (CR-42)	changes the VSIM local directory to the specified directory
change (CR-43)	modifies the value of a VHDL variable or Verilog register variable
change_menu_cmd (CR-44)	changes the command to be executed for a specified menu item label, in the specified menu, in the specified window
check contention add (CR-45)	enables contention checking for the specified nodes
check contention config (CR-46)	write checking messages to a file
check contention off (CR-47)	disables contention checking for the specified nodes
check float add (CR-48)	enables float checking for the specified nodes
check float config (CR-49)	write checking messages to a file
check float off (CR-50)	disables float checking for the specified nodes
check stable off (CR-52)	disables stability checking
check stable on (CR-51)	enables stability checking on the entire design
checkpoint (CR-53)	saves the state of your simulation
configure (CR-54)	invokes the List or Wave widget configure command for the current default List or Wave window
coverage clear (CR-59)	clears all coverage data obtained during previous run commands
coverage reload (CR-60)	seeds the coverage statistics with the output of a previous **coverage report** command
coverage report (CR-61)	produces a textual output of the coverage statistics that have been gathered up to this point
delete (CR-62)	removes HDL items from either the List or Wave window
describe (CR-63)	displays information about the specified HDL item

Command name	Action
disable_menu (CR-65)	disables the specified menu within the specified window
disable_menuitem (CR-66)	disables a specified menu item within the specified menu_path of the specified window
disablebp (CR-64)	temporarily turns off all existing breakpoints
do (CR-67)	executes commands contained in a macro file
down | up (CR-70)	searches for signal transitions or values in the specified List window
drivers (CR-73)	displays in the Main window the current value and scheduled future values for all the drivers of a specified VHDL signal or Verilog net
dumplog64 (CR-74)	dumps the contents of the *vsim.wav* file in a readable format
echo (CR-75)	displays a specified message in the Main window
edit (CR-76)	invokes the editor specified by the EDITOR environment variable
enable_menu (CR-78)	enables a previously-disabled menu
enable_menuitem (CR-79)	enables a previously-disabled menu item
enablebp (CR-77)	turns on all breakpoints turned off by the **disablebp** command (CR-64)
environment (CR-80)	display or change the current region/signal environmen
examine (CR-81)	examines one or more HDL items, and displays current values (or the values at a specified previous time) in the Main window
exit (CR-84)	exits the simulator and the Model*Sim* application
find (CR-85)	displays the full pathnames of all HDL items in the design whose names match the name specification you provide
force (CR-87)	allows you to apply stimulus to VHDL signals and Verilog nets interactively
getactivecursortime (CR-90)	gets the time of the active cursor in the Wave window
getactivemarkertime (CR-91)	gets the time of the active marker in the List window
lecho (CR-92)	takes one or more Tcl lists as arguments and pretty-prints them to the Transcript window

Command name	Action
log (CR-93)	creates a logfile containing simulation data for all HDL items whose names match the provided specifications
lshift (CR-95)	takes a Tcl list as argument and shifts it in-place one place to the left, eliminating the 0th element
lsublist (CR-96)	returns a sublist of the specified Tcl list that matches the specified Tcl glob pattern
macro_option (CR-97)	controls the speed and delay of macro (DO file) playback, plus the level of debugging feedback
modelsim (CR-98)	starts the Model*Sim* GUI without prompting you to load a design; valid only for Windows platforms
next (CR-100)	see "search and next" (CR-141)
noforce (CR-101)	removes the effect of any active **force** (CR-87) commands on the selected HDL items
nolog (CR-102)	suspends writing of data to the logfile for the specified signals
notepad (CR-104)	calls up a simple text editor
nowhen (CR-105)	deactivates selected **when** (CR-226) commands
onbreak (CR-106)	used within a macro; it specifies a command to be executed when running a macro that encounters a breakpoint in the source code
onElabError (CR-107)	specifies one or more commands to be executed when an error is encountered during elaboration
onerror (CR-108)	used within a macro; it specifies one or more commands to be executed when a running macro encounters an error
pause (CR-109)	placed within a macro; interrupts the execution of that macro
play (CR-110)	plays a sequence of keyboard and mouse actions, which were previously saved to a file with the **record** command (CR-130)
power add (CR-111)	specifies the signals or nets to track for power information
power report (CR-112)	writes out the power information for the specified signals or nets

Command name	Action
power reset (CR-113)	selectively resets power information to zero for the signals or nets specified with the **power add** command (CR-111)
printenv (CR-114)	echoes to the Transcript window the current names and values of all environment variables
profile clear (CR-115)	clear any data that has been gathered during previous **run** commands
profile interval (CR-116)	select the frequency with which the profiler collects samples during a run command
profile off (CR-117)	discontinue runtime profiling
profile on (CR-118)	enable runtime analysis of where your simulation is spending its time
profile option (CR-119)	allows various profiling options to be changed
profile report (CR-120)	produce a textual output of the profiling statistics that have been gathered up to this point
project (CR-121)	performs common operations on new projects
property list (CR-123)	changes one or more properties of the specified signal, net or register in the List window (10-174)
property wave (CR-124)	changes one or more properties of the specified signal, net or register in the Wave window (10-212)
pwd (CR-126)	displays the current directory path in the Main transcript window
quietly (CR-127)	turns off transcript echoing for the specified command
quit (CR-128)	exits the simulator
radix (CR-129)	specifies the default radix to be used
record (CR-130)	starts recording a replayable trace of all keyboard and mouse actions
report (CR-131)	displays the value of all simulator control variables, or the value of any simulator state variables relevant to the current simulation
restart (CR-133)	reloads the design elements and resets the simulation time to zero

Command name	Action
restore (CR-134)	restores the state of a simulation that was saved with a **checkpoint** command (CR-53) during the current invocation of VSIM
resume (CR-135)	resume execution of a macro file after a **pause** command (CR-109), or a breakpoint
right | left (CR-136)	searches right (next) or left (previous) for signal transitions or values in the specified Wave window
run (CR-139)	advances the simulation by the specified number of timesteps
search and next (CR-141)	search the specified window for one or more items matching the specified pattern(s)
searchLog (CR-144)	searches one or more of the currently open logfiles for a specified condition
seetime (CR-146)	scrolls the List or Wave window to make the specified time visible
shift (CR-147)	shifts macro parameter values down one place
show (CR-148)	lists HDL items and subregions visible from the current environment
splitio (CR-149)	operates on a VHDL inout or out port to create a new signal having the same name as the port suffixed with "__o"
status (CR-150)	lists all current interrupted macros
step (CR-151)	steps to the next HDL statement
stop (CR-152)	used with the **when** command (CR-226) to stop simulation in batch files
tb (CR-153)	displays a stack trace for the current process in the Main transcript window
toggle add (CR-154)	enables collection of toggle statistics for the specified nodes
toggle report (CR-156)	displays to the screen a list of all nodes that have not transitioned to both 0 and 1 at least once
toggle reset (CR-155)	resets the toggle counts to zero for the specified nodes

Command name	Action
transcribe (CR-157)	displays a command in the Main window, then executes the command
transcript (CR-158)	controls echoing of commands executed in a macro file; also works at top level in batch mode
tssi2mti (CR-159)	converts a vector file in Summit Design Standard Events Format into a sequence of VSIM **force** (CR-87) and **run** (CR-139) commands
vcd add (CR-160)	adds the specified items to the VCD file
vcd checkpoint (CR-161)	dumps the current values of all VCD variables to the VCD file
vcd comment (CR-162)	inserts the specified comment in the VCD file
vcd file (CR-163)	specifies the filename and state mapping for the VCD file created by a **vcd add** command (CR-160)
vcd flush (CR-165)	flushes the contents of the VCD file buffer to the VCD file
vcd limit (CR-166)	specifies the maximum size of the VCD file
vcd off (CR-167)	turns off VCD dumping and records all VCD variable values as x
vcd on (CR-168)	turns on VCD dumping and records the current values of all VCD variables
vcom (CR-169)	compiles VHDL design units
vdel (CR-175)	deletes a design unit from a specified library
vdir (CR-177)	selectively lists the contents of a design library.
vgencomp (CR-178)	writes a Verilog module's equivalent VHDL component declaration to standard output; invoked from DOS prompt only
view (CR-180)	opens a Model*Sim* window and brings it to the front of the display
virtual delete|describe|define (CR-182)	**delete** removes the matching virtuals; **describe** prints a complete description of the data type of one or more virtual signals; **define** prints the definition of the virtual signal or function in the form of a command that can be used to re-create the object
virtual expand (CR-183)	produces a list of all the non-virtual objects contained in the virtual signal(s)

Command name	Action
virtual function (CR-184)	creates a new signal that consists of logical operations on existing signals and simulation time
virtual hide|nohide (CR-188)	**hide** sets a flag in the specified real or virtual signals so that the signals do not appear in the Signals window; **nohide** resets the flag
virtual log|nolog (CR-189)	**log** causes the sim-mode dependent signals of the specified virtual signals to be logged by the kernel; **nolog** causes the specified virtual signals to be un-logged by the kernel
virtual region (CR-190)	creates a new user-defined design hierarchy region
virtual save (CR-191)	saves the definitions of virtuals to a file
virtual show|count (CR-192)	**show** lists the full path names of all the virtuals explicitly defined; **count** counts the number of explicitly declared virtuals that have not been saved and that were not read in using a macro file
virtual signal (CR-193)	creates a new signal that consists of concatenations of signals and subelements
vlib (CR-198)	selectively lists the contents of a design library
vlib (CR-198)	creates a design library
vlog (CR-199)	compiles Verilog design units
vmake (CR-205)	use a UNIX or Windows MAKE program to maintain libraries; invoked from UNIX or DOS prompt only
vmap (CR-207)	defines a mapping between a logical library name and a directory by modifying the *modelsim.ini* file
vsim (CR-208)	loads a new design into the simulator
vsim<info> (CR-219)	returns information about the current VSIM executable
vsource (CR-220)	displays an HDL source file in the VSIM Source window
wlf2log (CR-223)	translates a Model*Sim* logfile (*vsim.wav*) to a QuickSim II logfile
when (CR-226)	instruct VSIM to perform actions when the specified conditions are met
where (CR-230)	displays information about the system environment

Command name	Action
write format (CR-231)	records the names and display options of the HDL items currently being displayed in the List or Wave window
write list (CR-232)	records the contents of the most recently opened, or specified List window in a list output file
write preferences (CR-233)	saves the current GUI preference settings to a Tcl preference file
write report (CR-234)	prints a summary of the design being simulated
write transcript (CR-235)	writes the contents of the Main window transcript to the specified file
write tssi (CR-236)	ecords the contents of the default or specified List window in a “TSSI format” file
write wave (CR-238)	records the contents of the most currently opened, or specified Wave window in PostScript format

abort

The **abort** command halts the execution of a macro file interrupted by a breakpoint or error. When macros are nested, you may choose to abort the last macro only, abort a specified number of nesting levels, or abort all macros. The **abort** command may be used within a macro to return early.

Syntax

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

Arguments

None.

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

See also

onbreak command (CR-106), **restore** command (CR-134), and the **run** command (CR-139)

add button

The **add button** command adds a user-defined button to the Main window button bar. New buttons are added to the right end of the bar. You can also add buttons with a Model*Sim* tool: "The Button Adder" (10-265).

Returns the path name of the button widget created.

Syntax

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

Arguments

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

`<Cmd>`
The command to be executed when the button is clicked with the left mouse button. To echo the command and display the return value in the Main window, prefix the command with the **transcribe** command (CR-157). **Transcribe** will also echo the results to the transcript window. Required.

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

Note: The **transcribe** command (CR-157) will not work when the simulator is running, therefore don't use **transcribe** with NoDisable.

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

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

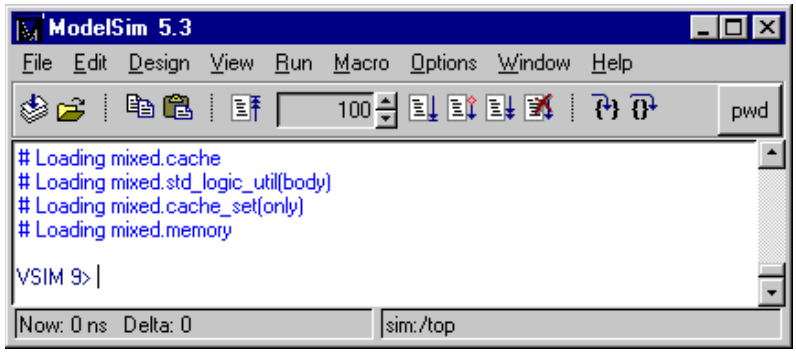
```
.controls.button_7 config
```

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

Examples

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

Creates a button labeled "pwd" that invokes the **transcribe** command (CR-157) with the **pwd** Tcl command, and echoes the command and its results to the Main window. The button remains active during a run.



The pwd button example is available in the following DO file: *<install_dir>/modeltech/examples/addbutton.do*. You can run the DO file to add the pwd button shown in the illustration, or modify the file for different results.

To execute the DO file, select **Macro > Execute Macro** from the Main window menu bar, or use the **do** command (CR-67) from the Model*Sim* command line.

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

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

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

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

```
.controls.button_7 config -command {run 10000} -bg red
```

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

See also

transcribe command (CR-157), and the "The Button Adder" (10-265) tool

add list

The **add list** command lists VHDL variables, and Verilog nets and registers, and their values in the List window. If no port mode is specified, all interface items and internal items are listed. Without arguments, the command displays the List window. The **add list** command also allows specification of user-defined buses.

Syntax

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

Arguments

`-strobe, -collapse, -delta, -nodelta`
These options are no longer part of the **add list** (formerly the **list** command). Use the **configure list** command instead. The command equivalents for the -collapse, -delta, and -nodelta options are shown below; see the **configure** command (CR-54) for more detail.

5.0 or newer	4.6x
configure list -delta collapse	list -collapse
configure list -delta all	list -delta
configure list -delta none	list -nodelta

`-window <wname>`
Adds HDL items to the specified List window <wname> (i.e., list2). Optional. Used to specify a particular window when multiple instances of that window type exist. Selects an existing window; does not create a new window. Use the **view** command (CR-180) with the **-new** option to create a new window.

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

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

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

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

`-internal`
For use with wild card searches. Specifies that the scope of the search is to include internal items if they match the item_name specification. Optional.

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

`-<radix>`
Specifies the radix for the items that follow in the command. Optional. Valid entries (or unique abbreviations) are:

```
binary
octal
decimal (or signed)
unsigned
hexadecimal
ascii
```

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

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

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

`-trigger`
Specifies that items are to be listed and causes the List window to be updated when the item changes. Optional. This switch is the default.

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

`-label <name>`
Specifies an alternative signal name to be displayed as a column heading in the listing. Optional. This alternative name is not valid in a **force** (CR-87) or **examine** (CR-81) command, however. It can optionally be used in a **search and next** command (CR-141) with the **list** option.

`<item_name>`
Specifies the name of the item to be listed. Optional. Wildcard characters are allowed. Variables may be added if preceded by the process name. For example,

```
add list myproc/int1
```

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

```
-keep
```

The original specified items are not removed; otherwise they are removed.

Note: You can use the **Edit > Combine** selection from the "List window" (10-174) menu to create a user-defined bus.

Examples

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

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

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

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

`add list clk -not a b c d`
Lists clk, a, b, c, and d only when clk changes.

`add list -not clk a b c d`
Lists clk, a, b, c, and d every 100 time units.

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

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

See also

log command (CR-93)

add_menu

The **add_menu** command adds a menu to the menu bar of the specified window, using the specified menu name. The menu may be justified to the left or right side of the menu bar. Use the **add_menuitem** (CR-30), **add_separator** (CR-31), **add_menucb** (CR-28), and **add_submenu** (CR-32) commands to complete the menu.

Returns the full Tk pathname of the new menu.

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

Syntax

```
add_menu
      <window_name> <menu_name> [<shortcut>] [<side>]
```

Arguments

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

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

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

`<side>`
Justify the menu "left" or "right". Optional. Default is "left".

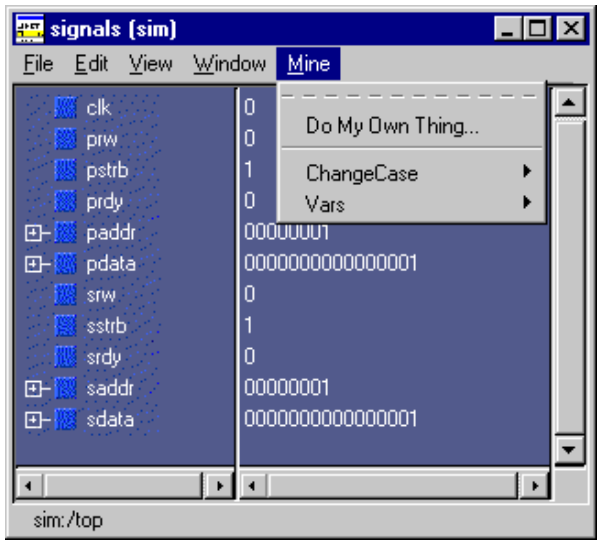
Examples

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

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

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

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



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

To execute the DO file, select **Macro** > **Execute Macro** from the Main window menu bar, or use the **do** command (CR-67) from the Model*Sim* command line.

See also

add_menucb command (CR-28), **add_menuitem** command (CR-30), **add_separator** command (CR-31), **add_submenu** command (CR-32), and the **change_menu_cmd** command (CR-44)

add_menucb

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

Syntax

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

Arguments

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

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

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

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

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

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

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

Examples

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

See also

add_menu command (CR-26), **add_menuitem** command (CR-30), **add_separator** command (CR-31), **add_submenu** command (CR-32), and the **change_menu_cmd** command (CR-44)

The **add_menucb** command is also used as part of the **add_menu** (CR-26) example.

add_menuitem

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

Syntax

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

Arguments

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

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

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

`<Cmd>`
The command to be executed when the menu item is selected with the left mouse button. To echo the command and display the return value in the Main window, prefix the command with the **transcribe** command (CR-157). **Transcribe** will also echo the results to the transcript window. Required.

`<accel_key>`
A number, starting from 0, of the letter to underline in the menu text. Used to indicate a keyboard shortcut key. Optional. Default is to pick the first character in <Text> that is unique across the menu items within the menu.

If accel_key = 1, no letter is underlined and no acceleration key will be defined.

Examples

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

See also

add_menu command (CR-26), **add_menucb** command (CR-28), **add_separator** command (CR-31), **add_submenu** command (CR-32), and the **change_menu_cmd** command (CR-44)

The **add_menuitem** command is also used as part of the **add_menu** (CR-26) example.

add_separator

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

Syntax

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

Arguments

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

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

Examples

```
add_separator $wname user
```

See also

add_menu command (CR-26), **add_menucb** command (CR-28), **add_menuitem** command (CR-30), **add_submenu** command (CR-32), and the **change_menu_cmd** command (CR-44)

The **add_separator** command is also used as part of the **add_menu** (CR-26) example.

add_submenu

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

Returns the full Tk path to the new submenu widget.

Syntax

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

Arguments

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

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

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

`<accel_key>`
A number, starting from 0, of the letter to underline in the menu text. Used to indicate a keyboard shortcut key. Optional. Default is to pick the first character in <Text> that is unique across the menu items within the submenu.

If accel_key = -1, no letter is underlined and no acceleration key will be defined.

See also

add_menu command (CR-26), **add_menucb** command (CR-28), **add_menuitem** command (CR-30), **add_separator** command (CR-31), and the **change_menu_cmd** command (CR-44)

The **add_submenu** command is also used as part of the **add_menu** (CR-26) example.

add wave

The **add wave** command adds VHDL signals, and Verilog nets and registers to the Wave window. It also allows specification of user defined buses.

Syntax

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

Arguments

`-window <wname>`
Adds HDL items to the specified window <wname> (i.e., wave2). Optional. Used to specify a particular window when multiple instances of that window type exist. Selects an existing window; does not create a new window. Use the **view** command (CR-180) with the **-new** option to create a new window.

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

`-expandall <signal_name>`
Causes a compound signal to be fully expanded immediately. Optional. The <signal_name> is required, and may include wildcards.

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

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

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

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

`-internal`
For use with wild card searches. Specifies that the scope of the search is to include internal items if they match the item_name specification. Optional.

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

`-<radix>`
Specifies the radix for the items that follow in the command. Optional. Valid entries (or any unique abbreviation) are:

```
binary
octal
decimal (or signed)
unsigned
hexadecimal
symbolic
ascii
```

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

If you specify a radix for an array of a VHDL enumerated type, Model*Sim* converts each signal value to 1, 0, Z, or X. See also, "Preference variables located in TCL files" (B-406).

`-<format>`
Specifies type of items:

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

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

The way each state is displayed is specified by the logic type display preferences, see "Preference variables located in INI and MPF files" (B-394). Analog signals are sized by **-scale** and by **-offset**. Analog-step changes to the new time before plotting the new Y. Analog-interpolated draws a diagonal line. Analog-backstep plots the new Y before moving to the new time. See "Editing and formatting HDL items in the Wave window" (10-226).

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

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

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

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

The position and size of the waveform is given by:

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

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

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

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

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

This alternative name is not valid in a **force** (CR-87) or **examine** (CR-81) command, however. It can optionally be used in a **search and next** command (CR-141) with the **wave** option.

`<item_name>`
Specifies the names of HDL items to be included in the Wave window display. Optional. Wildcard characters are allowed. Variables may be added if preceded by the process name. For example,

```
add list myproc/int1
```

`{<item_name> [-flatten] {sig1 sig2 sig3 ...}}`
Creates a user-defined bus in place of <item_name>; 'sig*i*' are signals to be concatenated within the user-defined bus. Optional. The following option is available:

`-flatten`
Creates an array signal that cannot be expanded to show waveforms of

individual elements. Gives greater flexibility in the types of elements that can be combined, but loses the original element names. Without the **-flatten** option, all items must be either all scalars or all arrays of the same size. With **-flatten**, specified items may be either scalars or various sized arrays as long as they have the same element enumeration type.

Note: You can use the **Edit > Combine** selection from the "Wave window" (10-212) menu to create a user-defined bus.

Examples

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

Displays an item named *out2*. The item is specified as being a logic item presented in gold.

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

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

alias

The **alias** command creates a new Tcl procedure that evaluates the specified commands. Used to create a user-defined alias. Any arguments passed on invocation of the alias will be passed through to the specified commands. Returns nothing.

Syntax

```
alias
      <aka> "<cmds>"
```

Arguments

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

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

Examples

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

Creates a Tcl procedure, "myquit", that when executed, writes the contents of the List window to the file *mylist.save* by invoking **write list** (CR-232), and quits Model*Sim* by invoking **quit** (CR-128).

batch_mode

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

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 also use the **batch_mode** command to determine which command to use. For example:

```
if [batch_mode] {

    log /*

} else {

    add wave /*

}
```

See also

"Running command-line and batch-mode simulations" (E-440)

bd

The **bd** command deletes a breakpoint.

Syntax

```
bd
      <filename> <line_number>
```

Arguments

`<filename>`
Specifies the name of the source file in which the breakpoint is to be deleted. Required. The filename must match the one used to previously set the breakpoint, including whether a full pathname or a relative name was used.

`<line_number>`
Specifies the line number of the breakpoint to be deleted. Required.

Examples

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

See also

bp command (CR-40), and the **onbreak** command (CR-106)

bp

The **bp** or breakpoint command sets a breakpoint. If the source file name and line number are omitted, or the -query option is used, this command lists all the breakpoints that are currently set. Otherwise, the command sets a breakpoint in the specified file at the specified line. Once set, the breakpoint affects every instance in the design.

Syntax

```
bp
      [-query <filename>] [<filename>] [<line_number> [{<command>...}]]
```

Arguments

`-query<filename>`
Returns a list of the currently set breakpoints. Optional.

`<filename>`
Specifies the name of the source file in which the breakpoint is to be set. Required when **-query** is used; otherwise, it is optional. If omitted, all current breakpoints are listed.

`<line_number>`
Specifies the line number at which the breakpoint is to be set. Optional; if omitted, all current breakpoints are listed.

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

Any commands that follow a **run** (CR-139), or **step** (CR-151) command will be ignored. The **run**, or **step** command terminates the breakpoint sequence. This applies if macros are used with the **bp** command string as well. A **restore** (CR-134) command should not be used.

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

Examples

`bp`

Lists all existing breakpoints in the design, together with the source file names and any commands that have been assigned to breakpoints.

`bp -query testadd.vhd`

Lists the line number of all breakpoints in testadd.vhd.

`bp alu.vhd 147`

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

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

Executes the *macro.do* macro file after the breakpoint.

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

Sets a breakpoint at line 22 of the file *test.vhd* and examines the values of the two variables var1 and var2.

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

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

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

See also

add button command (CR-20), **bd** command (CR-39), **disablebp** command (CR-64), **enablebp** command (CR-77), and the **onbreak** command (CR-106), and the **when** command (CR-226)

cd

The Tcl **cd** command changes the VSIM local directory to the specified directory. See the Tcl man pages (Main window: **Help > Tcl Man Pages**) for any **cd** command options. Returns nothing.

Syntax

```
cd
      <dir>
```

Description

After you change the directory with **cd**, VSIM continues to write the *vsim.wav* file in the directory where the first **add wave** (CR-33), **add list** (CR-22) or **log** (CR-93) command was executed. After completing simulation of one design, you can use the **cd** command to change to a new design, then use the **vsim** command (CR-208) to load a new design.

Use the **where** command (CR-230) or the Tcl **pwd** command (see the Tcl man pages, Main window: **Help > Tcl Man Pages**) to confirm the current directory.

See also

where command (CR-230), **vsim** command (CR-208), and the Tcl man page for the **cd**, **pwd** and **exec** commands

change

The **change** command modifies the value of a VHDL variable or Verilog register variable. The simulator must be at a breakpoint or paused after a **step** command (CR-151) to change a VHDL variable.

Syntax

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

Arguments

`<variable>`
Specifies the name of a variable. Required. The variable 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.

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

Examples

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

See also

force command (CR-87)

change_menu_cmd

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

Syntax

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

Arguments

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

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

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

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

See also

add_menu command (CR-26), **add_menucb** command (CR-28), **add_menuitem** command (CR-30), **add_separator** command (CR-31), and the **add_submenu** command (CR-32)

check contention add

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

Syntax

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

Arguments

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

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

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

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

`-internal`
Enables checking on internal items. Optional.

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

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

See also

"Bus contention checking" (E-446)

check contention config

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

Syntax

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

Arguments

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

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

See also

"Bus contention checking" (E-446)

check contention off

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

Syntax

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

Arguments

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

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

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

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

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

`-internal`
Disables checking on internal items. Optional.

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

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

See also

"Bus contention checking" (E-446)

check float add

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

Syntax

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

Arguments

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

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

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

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

`-internal`
Enables checking on internal items. Optional.

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

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

See also

"Bus float checking" (E-447)

check float config

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

Syntax

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

Arguments

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

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

See also

"Bus float checking" (E-447)

check float off

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

Syntax

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

Arguments

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

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

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

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

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

`-internal`
Disables checking on internal items. Optional.

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

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

See also

Bus float checking (E-447)

check stable on

The **check stable on** command enables stability checking on the entire design. Design stability checking detects when circuit activity has not settled within a user-defined period for synchronous designs.

Syntax

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

Arguments

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

`-period <time>`
Specifies the clock period (which is assumed to begin at the time the **check stable on** command is issued). Optional. This option is required the first time you invoke the **check stable on** command. It is not required if you later enable checking after it was disabled with the **check stable off** command (CR-52). See the **check stable off** command (CR-52).

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

Examples

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

See also

"Design stability checking" (E-447)

check stable off

The **check stable off** command disables stability checking. You may later enable it with **check stable on** (CR-51), and meanwhile, the clock cycle numbers and boundaries are still tracked. See the **check stable on** command (CR-51).

Syntax

```
check stable off
```

Arguments

None.

See also

"Design stability checking" (E-447)

checkpoint

The **checkpoint** command saves the state of your simulation. The **checkpoint** command saves the simulation kernel state, the *vsim.wav* file, the list of the HDL items shown in the List and Wave windows, the file pointer positions for files opened under VHDL and the Verilog **$fopen** system task, and the states of foreign architectures. Changes you made interactively while running VSIM are not saved; for example, VSIM macros, command-line interface additions like user-defined commands, and states of graphical user interface windows are not saved.

Once saved, a checkpoint file may be used with the **restore** command (CR-134) during the same simulation to restore the simulation to a previous state. A VSIM session may also be started with a checkpoint file by using the **vsim -restore** command (CR-208).

Syntax

```
checkpoint
      <filename>
```

Arguments

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

See also

restore command (CR-134), **restart** command (CR-133), **vsim** command (CR-208), and see "The difference between checkpoint/restore and restarting" (E-439) for a discussion of the difference between **restart** and **checkpoint/restore**.

configure

The **configure** (**config**) command invokes the List or Wave widget configure command for the current default List or Wave window. To change the default window, use the **view** command (CR-180).

Returns the values of all attributes if no options, or the value of one attribute when one option and no value.

Syntax

```
configure
      list|wave [-window <wname>] [<option> <value>]
      [-delta [all | collapse | none]] [-gateduration <duration_open>]
      [-gateexpr {<expression>]] [-usegating] [-strobeperiod]
      [-strobestart] [-usesignaltriggers] [-usestrobe] [-timecolor]
      [-vectorcolor] [-gridcolor]
```

Arguments

`list|wave`
Specifies either the List or Wave widget to configure. Required.

`-window <wname>`
Specifies the name of the List or Wave window to target for the **configure** command (the **view** command (CR-180) allows you to create more than one List or Wave window). Optional. If no window is specified the default window is used; the default window is determined by the most recent invocation of the **view** command (CR-180).

`<option> <value>`

`-bg`
Specifies the window background color. Optional.

`-fg`
Specifies the window foreground color. Optional.

`-selectbackground`
Specifies the window background color when selected. Optional.

`-selectforeground`
Specifies the window foreground color when selected. Optional.

`-font`
Specifies the font used in the widget. Optional.

`-height`
Specifies the height in pixels of each row. Optional.

Arguments, List window only

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

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

`-gateexpr {<expression>]`
Specifies the expression for trigger gating. The expression is evaluated when the List window would normally have displayed a row of data. Optional. See the "GUI_expression_format" (CR-250) for information on expression syntax.

`-usegating`
Enables triggers to be gated on (a value of 1) or off (a value of 0) by an overriding expression. The default is off. See "Setting List window display properties" (10-177) for additional information on using gating with triggers.

`-strobeperiod`
Specifies the period of the list strobe. (When using a time unit, the time value and unit must be placed in curly brackets.) Optional.

`-strobestart`
Specifies the start time of the list strobe. Optional.

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

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

Arguments, Wave window only

`-timecolor`
Specifies the time axis color; the default is green. Optional.

`-vectorcolor`
Specifies the vector waveform color; the default is yellow. Optional.

`-gridcolor`
Specifies the background grid color; the default is grey50. Optional.

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

Note: To get a more readable listing of all attributes and current values, use the **lecho** (CR-92) command, which pretty-prints a Tcl list (see example below).

```
lecho [config list]
```

Returns a pretty-printed list of the List window attributes and current values as shown below:

```
VSIM 46> lecho [config list]
# {
#    {
#       -adjustdividercommand adjustdividercommand Command {} VListPlaceDivider
#    }
#    {
#       -background background Background {light blue}
```

```
#       #d9dfff
#    }
#    {-bd borderWidth}
#    {-bg background}
#    {-borderwidth borderWidth BorderWidth 2 2}
#    {
#       -cursor cursor Cursor {} {}
#    }
#    {-busycursor busycursor BusyCursor watch watch}
#    {-delta delta Delta all all}
#    {
#       -timeunits timeunits TimeUnits {} ns
#    }
#    {-fastload fastload FastLoad 0 0}
#    {-fastscroll fastscroll FastScroll 0 0}
#    {-foreground foreground Foreground black Black}
#    {-fg foreground}
#    {-fixwidth fixwidth Width 52 130}
#    {-fixoffset fixoffset Offset 0 0}
#    {-font font Font *courier-medium-r-normal-*-12-* -adobe-courier-medium-r-
     normal--12-*-*-*-*-*}
#    {-h height}
#    {-height height Height 15 187}
#    {-highlightcolor highlightColor HighlightColor white white}
#    {-highlightthickness highlightThickness HighlightThickness 2 2}
#    {
#       -logfile logfile Filename {} vsim.wav
#    }
#    {-namelimit namelimit NameLimit 5 5}
#    {-relief relief Relief ridge ridge}
#    {-selectbackground selectBackground Background grey20 Blue}
#    {-selectforeground selectForeground Foreground grey20 White}
#    {-selectwidth selectWidth BorderWidth 2 2}
#    {-shortnames shortnames ShortNames 0 0}
#    {-signalnamewidth signalnamewidth SignalNameWidth 0 0}
#    {
#       -strobeperiod strobePeriod StrobeTime {0 ns}
#       {0 ns}
#    }
#    {
#       -strobestart strobeStart StrobeTime {0 ns}
#       {0 ns}
#    }
#    {-usesignaltriggers usesignaltriggers UseSignalTriggers 1 1}
#    {-usestrobe usestrobe UseStrobe 0 0}
#    {-w width}
#    {-width width Width 200 363}
#    {
#       -xscrollcommand xscrollcommand Command {} {.list.xscroll set}
#    }
```

```
#    {
#       -yscrollcommand yscrollcommand Command {} {.list.yscroll set}
#    }
# }
VSIM 47>
```

Examples

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

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

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

See also

view command (CR-180)

coverage clear

The **coverage clear** command is used to clear all coverage data obtained during previous run commands. After this command is executed all line number execution count data will be reset.

Syntax

```
coverage clear
```

Arguments

None.

See also

coverage reload command (CR-60), **coverage report** command (CR-61)

coverage reload

The **coverage reload** command is used to seed the coverage statistics with the output of a previous **coverage report** command. This allows you (for example) to gather statistics from multiple simulation runs into a single report.

Syntax

```
coverage reload
      <filename> [ -keep ]
```

Arguments

`<filename>`

Specifies the file containing data to reload. Required. This file should be the output of a previous COVERAGE REPORT -lines command.

`-keep`

By default, source files listed in the file being reloaded that do NOT exist in the current design will have their coverage data discarded.

By specifying the -keep option, the data will be kept, even though it does not correspond to any file or line in the current design.

The **coverage reload** command allows the accumulation of coverage statistics for multiple simulation invocations.

By doing a **coverage report -lines** at the end of each simulation, and then a **coverage reload -keep** right at the start of each subsequent invocation of the simulator, one can accumulate coverage data for a suite of different designs.

See also

coverage clear command (CR-59), **coverage report** command (CR-61)

coverage report

The **coverage report** command is used to produce a textual output of the coverage statistics that have been gathered up to this point. (The **View** > **Other** > **Source Coverage** menu pick allows you to view this data more interactively.)

Syntax

```
coverage report
      [ -file <filename> ] [ -lines ]
```

Arguments

`-file <filename>`
Specifies a file name for the report. Optional. Default is to write report to the Transcript Window.

`-lines`
Specifies whether to report coverage information for individual lines as well as summarizing on a per file basis. Optional. Default is to only summarize by files.

If you are intending to use the **coverage reload** command, you should specify this switch.

The report is given in tabular format. For each source file that had design elements simulated, a summary consisting of file name, number of executable lines, number of lines that were actually executed and a percentage of coverage are given.

If the -lines switch is specified then following each file summary line is a list of executable lines in that file, along with how many times that line was executed.

See also

coverage clear command (CR-59), **coverage reload** command (CR-60)

delete

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

Syntax

```
delete
      list | wave [-window <wname>] <item_name>
```

Arguments

`list | wave`
Specifies the target window for the **delete** command. Required.

`-window <wname>`
Specifies the name of the List or Wave window to target for the **delete** command (the **view** command (CR-180) allows you to create more than one List or Wave window). Optional. If no window is specified the default window is used; the default window is determined by the most recent invocation of the **view** command (CR-180).

`<item_name>`
Specifies the name of an item. Required. Must match the item name used in the **add list** (CR-22) or **add wave** (CR-33) command. Multiple item names may be specified. Wildcard characters are allowed.

Examples

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

See also

add list command (CR-22), **add wave** command (CR-33), and "Wildcard characters" (CR-248)

describe

The **describe** command displays information about the specified HDL item. The description is displayed in the Main window (10-158). The following kinds of items can be described:

- **VHDL**
  signals, variables, and constants
- **Verilog**
  nets and registers

All but VHDL variables and constants may be specified as hierarchical names. VHDL variables and constants can be described only when visible from the current process that is either selected in the Process window or is the currently executing process (at a breakpoint for example).

Syntax

```
describe
      <name>
```

Arguments

`<name>`
Specifies the name of an HDL item. Multiple names and wildcards are accepted. Required.

disablebp

The **disablebp** command temporarily turns off all existing breakpoints. To turn the breakpoints back on again, use the **enablebp** command (CR-77).

Syntax

```
disablebp
```

Arguments

None.

See also

bd command (CR-39), **bp** command (CR-40), **onbreak** command (CR-106), and the **restore** command (CR-134)

disable_menu

The **disable_menu** command disables the specified menu within the specified window. The disabled menu will become grayed-out, and nonresponsive. Returns nothing.

Syntax

```
disable_menu
      <window_name> <menu_path>
```

Arguments

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

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

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

Examples

```
disable_menu "" file
```
Disables the file menu of the Main VSIM window.

```
disable_menu .mywindow file
```
Disables the file menu of the mywindow window.

See also

enable_menu command (CR-78)

disable_menuitem

The **disable_menuitem** command disables a specified menu item within the specified menu_path of the specified window. The menu item will become grayed-out, and nonresponsive. Returns nothing.

Syntax

```
disable_menuitem
      <window_name> <menu_path> <label>
```

Arguments

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

`<menu_path>`
Name of the Tk menu-widget path. The path may include a submenu as shown in the example below. Required.

`<label>`
Menu item text. Required.

Examples

```
disable_menuitem .mywindow file.save "Save Results As..."
```
This command locates the mywindow window, and disables the Save Results As... menu item in the save submenu of the file menu.

See also

enable_menuitem command (CR-79)

do

The **do** command executes commands contained in a macro file. A macro file can have any name and extension. An error encountered during the execution of a macro file causes its execution to be interrupted, unless an **onerror** command (CR-108), or **onbreak** command (CR-106) has specified the **resume** command (CR-135).

Syntax

```
do
      <filename> [<parameter_value>]
```

Arguments

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

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

`<parameter_value>`
Specifies values that are to be passed to the corresponding parameters $1 through $9 in the macro file. Optional. Multiple parameter values must be separated by spaces. If you specify fewer parameter values than the number of parameters used in the macro, the unspecified values are treated as empty strings in the macro.

Note that there is no limit on the number of parameters that can be passed to macros, but only nine values are visible at one time. You can use the **shift** command (CR-147) to see the other parameters. The argc (B-420) returns the number of parameters passed.

Examples

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

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

Using other VSIM commands with macros

If you are executing a macro (DO file) when your simulation hits a breakpoint or causes a run-time error, VSIM interrupts the macro and returns control to the command line, where the following commands may be useful. (Any other legal command may be executed as well.)

command	result
run (CR-139) -continue	continue as if the breakpoint had not been executed, completes the **run** command (CR-139) that was interrupted
restore (CR-134)	continue running the macro
onbreak (CR-106)	specify a command to run when you hit a breakpoint within a macro
onElabError (CR-107)	specify a command to run when an error is encountered during elaboration
onerror (CR-108)	specify a command to run when an error is encountered within a macro
status (CR-150)	get a traceback of nested macro calls when a macro is interrupted
abort (CR-19)	terminate a macro once the macro has been interrupted or paused
pause (CR-109)	cause the macro to be interrupted, the macro can be resumed by entering a **resume** command (CR-135) via the command line
toggle add (CR-154)	control echoing of macro commands to the Transcript window

Using the Tcl **source** command with DO files

Either the **do** command or Tcl **source** command can execute a DO file, but they behave differently.

With the **source** command, the DO file is executed exactly as if the commands in it were typed in by hand at the prompt. Each time a breakpoint is hit the Source window is updated to show the breakpoint. This behavior could be inconvenient with a large DO file containing many breakpoints.

When a **do** commmand is interrupted by an error or breakpoint, it does not update any windows, and keeps the DO file "locked". This keeps the Source window from flashing, scrolling, and moving the arrow when a complex DO file is executed. Typically an **onbreak resume** command is used to keep the macro running as it hits breakpoints. Add an **onbreak abort**, command to the DO file if you want to exit the macro and update the Source window.

See also

VSIM can search for DO files based on the path list specified by the DOPATH (B-391). A DO file can also be called by *modelsim.ini* at startup, see "Using a startup file" (B-404). The Main transcript can be saved as a macro, see the **write transcript** command (CR-235).

down | up

The **down | up** command searches for signal transitions or values in the specified List window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. A condition to search for may also be identified by an expression using the **-expr** command option. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a signal takes on a particular value, or an expression of multiple signals evaluates to true. Use the mouse to select the desired signal and click on the desired starting location using the left mouse button. Then issue the **down | up** command. (The **seetime** command (CR-146) can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

Syntax

```
down/up
      [-expr {<expression>}] [-falling] [-noglitch] [-rising]
      [-value <sig_value>] [-window <wname>] [<n>]
```

Arguments

`-expr {<expression>}`
The List window will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced List window. A signal may be specified either by its full path or by the shortcut label displayed in the List window.

See "GUI_expression_format" (CR-250) for the format of the expression. The expression must be placed within curly braces.

`-falling`
Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-noglitch`
Specifies that delta-width glitches are to be ignored. Optional.

`-rising`
Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-value <sig_value>`
Specify a value of the signal to match. Optional. Must be specified in the same radix that the selected signal is displayed. Case is ignored, but otherwise must be an exact string match -- don't-care bits are not yet implemented.

`-window <wname>`
Use this option to specify an instance of the List window that is not the default. Optional. Otherwise, the default List window is used. Use the **view** command (CR-180) to change the default window.

`<n>`
Specifies to find the nth match. Optional. If less than n are found, the number found is returned with a warning message, and the marker is positioned at the last match.

Examples

Returns 1 if a match is found, 0 if not. If the nth match is requested and only m are found,
m < n, then it returns m.

```
down -noglitch -value FF23
```
Finds the next time which the selected vector transitions to FF23, ignoring glitches.

```
up
```
Goes to the previous transition on the selected signal.

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the "GUI_expression_format" (CR-250) and can be built with the aid of the "The GUI Expression Builder" (10-272).

```
down -expr {clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)}
```
Searches down for an expression that evaluates to a boolean 1 when signal clk just changed from low to high and signal mystate is the enumeration reading and signal /top/u3/addr is equal to the specified 32-bit hex constant; otherwise is 0.

```
down -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
```
Searches down for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal /top/u3/adder equals hex ac.

```
down -expr {((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode == writing)}
```
Searches down for an expression that evaluates to a boolean 1 when logfile time is between

23 and 54 microseconds, and clock just changed from low to high and signal mode is enumeration writing.

See also

"GUI_expression_format" (CR-250), **view** command (CR-180), and the **seetime** command (CR-146)

drivers

The **drivers** command displays in the Main window the current value and scheduled future values for all the drivers of a specified VHDL signal or Verilog net. The driver list is expressed relative to the top most design signal/net connected to the specified signal/net. If the signal/net is a record or array, each sub-element is displayed individually. This command reveals the operation of transport and inertial delays and assists in debugging models.

Syntax

```
drivers
      <item_name>
```

Arguments

`<item_name>`
Specifies the name of the signal or net whose values are to be shown. Required. All signal or net types are valid. Multiple names and wild cards are accepted.

dumplog64

The **dumplog64** command dumps the contents of the *vsim.wav* file in a readable format.

Syntax

```
dumplog64
      <filename>
```

Arguments

`<filename>`
The name of the dump file created. Required.

echo

The **echo** command displays a specified message in the Main window.

Syntax

```
echo
      <text_string>
```

Arguments

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

Examples

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

If the current time is 1000 ns, this command produces the message:

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

If the quotes are omitted, all blank spaces of two or more are compressed into one space.

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

If the current value of counter is 21, this command produces the message:

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

echo can also use command substitution, such as:

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

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

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

edit

The **edit** command invokes the editor specified by the EDITOR environment variable.

Syntax

```
edit
      [<filename>]
```

Arguments

`<filename>`
Specifies the name of the file to edit. Optional. If the <filename> is omitted, the editor opens the current source file.

See also

notepad command (CR-104), and the EDITOR (B-391)

enablebp

The **enablebp** command turns on all breakpoints turned off by the **disablebp** command (CR-64).

Syntax

```
enablebp
```

Arguments

None.

See also

bd command (CR-39), **bp** command (CR-40), **onbreak** command (CR-106), and the **restore** command (CR-134)

enable_menu

The **enable_menu** command enables a previously-disabled menu. The menu will be changed from grayed-out to normal, and will become responsive. Returns nothing.

Syntax

```
enable_menu
      <window_name> <menu_path>
```

Arguments

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

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

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

Examples

```
enable_menu "" file
```
Enables the previously-disabled file menu of the Main VSIM window.

```
enable_menu .mywindow file
```
Enables the previously-disabled file menu of the mywindow window.

See also

disable_menu command (CR-65)

enable_menuitem

The **enable_menuitem** command enables a previously-disabled menu item. The menu item will be changed from grayed-out to normal, and will become responsive. Returns nothing.

Syntax

```
enable_menuitem
      <window_name> <menu_path> <label>
```

Arguments

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

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

`<menu_path>`
Name of the Tk menu-widget path. The path may include a submenu as shown in the example below. Required.

`<label>`
Menu item text. Required.

Examples

```
enable_menuitem .mywindow file.save "Save Results As..."
```
This command locates the mywindow window, and enables the previously-disabled Save Results As... menu item in the save submenu of the file menu.

See also

disable_menuitem command (CR-66)

environment

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

Syntax

```
environment
      [-dataset] [-nodataset] [<pathname>]
```

Arguments

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

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

`<pathname>`
Specifies the pathname to which the current region/signal environment is to be changed. Optional; if omitted, the command causes the pathname of the current region/signal environment to be displayed.

Multiple levels of a pathname must be separated by the character specified in the PathSeparator (B-400). A single path separator character can be entered to indicate the top level. Two dots (..) can be entered to move up one level.

Examples

`env`
Displays the pathname of the current region/signal environment.

`env ..`
Moves up one level in the design hierarchy.

`env blk1/u2`
Moves down two levels in the design hierarchy.

`env /`
Moves to the top level of the design hierarchy.

examine

The **examine**, or **exa** command, examines one or more HDL items, and displays current values (or the values at a specified previous time) in the Main window (10-158). It optionally can compute the value of an expression of one or more items.

The following items can be examined at any time:

- **VHDL**
  signals and process variables
- **Verilog**
  nets and register variables

To examine a VHDL variable, the simulator must be paused after a **step** command (CR-151), a breakpoint, or you can specify a process label with the name. 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).

Syntax

```
examine
      [-time <time>] [-context] [-delta <delta>] [-env <path>] [-expr
      <expression>] [-<radix>] [-value] [-name] <name>...
```

Arguments

`-time <time>`
Specifies the time value between 0 and $now for which to examine the items. If an expression is specified it will be evaluated at that time. Optional.

The item to be examined must have been logged using the **add list** command (CR-22); the **log** command (CR-93) is not sufficient.

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

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

If used, **-time** must be the first option.

`-context`
Passes the region to look for signal name. Optional.

`-delta<delta>`
Specifies a simulation cycle at the specified time from which to fetch the value. The default is to use the last delta of the time step. Optional.

`-env <path>`
Specifies a path to look for signal name. Optional.

`-expr <expression>`
An expression to be evaluated. Optional. If the **-time** argument is present, the expression will be evaluated at the specified time, otherwise it will be evaluated at the current simulation time. See "GUI_expression_format" (CR-250) for the format of the expression. The expression must be placed within curly braces.

The expression argument to the examine statement can only involve signals that have been logged in the List window. The signal may be specified by either the full path, or its shortcut name displayed in the List window (if one has been specified).

`-<radix>`
Specifies the radix for the items that follow in the command. Optional. Valid entries (including unique abbreviations) are:

```
binary
octal
decimal (default for integers) or signed
hexadecimal
unsigned
ascii
```

Entries may be truncated to any length, for example, -binary could be expressed as -b or -bin, etc.

`-name`
Add display of the name. Optional. Useful for wildcard patterns. Returns name(s) as a curly-bracket separated Tcl list. Default is **-value** behavior.

The **lecho** command (CR-92) will return a list in pretty-print format.

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

`<name>...`
Specifies the name of any HDL item. Required(though not used if the **-expr** option is used). All item types are allowed, except those of the type file. Multiple names and wildcards are accepted. To examine a VHDL variable you can add a process label to the name. For example (make certain to use two underscore characters):

```
exa line__36/i
```

Examples

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

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

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

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

Commands like **find** (CR-85) and **examine** return their results as a Tcl list (just a blank-separated list of strings). You can do things like:

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

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

examine -hex [find *]
```

Note: The Tcl variable array, $examine (), can also be used to return values. For example, $examine (/clk). You can also examine an item in the Source window (10-200) by selecting it with the right mouse button.

See also

"GUI_expression_format" (CR-250)

exit

The **exit** command exits the simulator and the Model*Sim* application.

Syntax

```
exit
      [-force]
```

Argument

`-force`
Quits without asking for confirmation. Optional; if this argument is omitted, Model*Sim* asks you for confirmation before exiting.

Note: If you want to stop the simulation using a **when** command (CR-226), you must use a **stop** command (CR-152) within your when statement. DO NOT use an **exit** commad or a **quit** command (CR-128). The **stop** command acts like a breakpoint at the time it is evaluated.

find

The **find** command displays the full pathnames of all HDL items in the design whose names match the name specification you provide. If no port mode is specified, all interface items and internal items are found (that is, all items of modes IN, OUT, INOUT, and INTERNAL).

Syntax

```
find
      [-recursive] [-in] [-out] [-inout]
      [-internal] [-ports] <item_name> ...
```

Arguments

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

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

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

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

`-internal`
Specifies that the scope of the search is to include internal items. Optional.

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

`<item_name> ...`
Specifies the name for which you want to search. Required. Multiple names and wildcard characters are allowed.

Examples

`find -r /*`
Finds all items in the entire design.

`find *`
Displays the names of all items in the current region.

Additional search options

To search for HDL items within a specific display window, use the **search and next** (CR-141) or the menu sequence: **Edit** > **Find ...**.

See also

"Wildcard characters" (CR-248)

force

The **force** command allows you to apply stimulus to VHDL signals and Verilog nets interactively. Since **force** commands (like all VSIM commands) can be included in a macro file, it is possible to create complex sequences of stimuli.

Forcing of Virtual signals (9-145) is supported if the number of bits correspond to the signal value; forcing of virtual functions is not supported.

Syntax

```
force
      [-freeze | -drive | -deposit] [-cancel <period>] [-repeat <period>]
      <item_name> <value> [<time>] [, <value> <time> ...]
```

Arguments

`-freeze`
Freezes the item at the specified value until it is forced again or until it is unforced with a **noforce** command (CR-101). Optional.

`-drive`
Attaches a driver to the item and drives the specified value until the item is forced again or until it is unforced with a **noforce** command (CR-101). Optional.

This option is illegal for unresolved signals.

`-deposit`
Sets the item to the specified value. The value remains until there is a subsequent driver transaction, or until the item is forced again, or until it is unforced with a **noforce** command (CR-101). Optional.

If one of the **-freeze**, **-drive**, or **-deposit** options is not used, then **-freeze** is the default for unresolved items and **-drive** is the default for resolved items.

If you prefer **-freeze** as the default for resolved and unresolved VHDL signals, you can change the default force kind in the *imodelsim.tcl* file (see "Preference variables located in INI and MPF files" (B-394)), or by using the DefaultForceKind (B-398).

`-cancel <period>`
Cancels the **force** command after the specified **<period>** of current time units. Cancellation occurs at the last simulation delta cycle of a time unit. A value of zero cancels the force at the end of the current time period. Optional.

`-repeat <period>`
Repeats the **force** command, where **<period>** is the amount of time of the repeat period. A repeating **force** command will force a value before other non-repeating **force** commands that occur in the same time step. Optional.

`<item_name>`
Specifies the name of the HDL item to be forced. Required. A wildcard is permitted only if it matches one item. See "HDL item pathnames" (CR-245) for the full syntax of an item name. The item name must specify a scalar type or a one-dimensional array of character enumeration. You may also specify a record sub-element, an indexed array, or a sliced array, as long as the type is one of the above. Required.

`<value>`
Specifies the value that the item is forced to. The specified value must be appropriate for the type. Required.

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

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

Note: For based numbers in VHDL, Model*Sim* converts each 1 or 0 in the value to one of the values in the enumerated type.This translation is specified by the force mapping preferences in the *imodelsim.tcl* file , see "Preference variables located in INI and MPF files" (B-394). If Model*Sim* cannot find a translation for 0 or 1, it uses the left bound of the signal type (type'left) for that value.

`<time>`
Specifies the time that the value is applied. The time is relative to the current time unless an absolute time is specified by preceding the value with the character @.

If the time units are not specified, then the default is the resolution units selected at simulation start-up. Optional.

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

Examples

```
force input1 0
```
Forces input1 to 0 at the current simulator time.

```
force bus1 01XZ 100 ns
```
Forces bus1 to 01XZ at 100 nanoseconds after the current simulator time.

```
force bus1 16#f @200
```
Forces bus1 to 16#F at the absolute time 200 measured in the resolution units selected at simulation start-up.

```
force input1 1 10, 0 20 -r 100
```
Forces input1 to 1 at 10 time units after the current simulator time and to 0 at 20 time units after the current simulation time. This repeats every 100 time units, so the next transition is to 1 at 110 time units afterwards.

```
force input1 1 10 ns, 0 {20 ns} -r 100ns
```
Similar to the previous example, but also specifies the time units. Time unit expressions preceding the "-r" must be placed in curly braces.

```
force s 1 0, 0 100 -repeat 200 -cancel 1000
```
Forces signal s to alternate between values 1 and 0 every 100 time units until time 1000. Cancellation occurs at the last simulation delta cycle of a time unit. So,

```
force s 1 0 -cancel 0
```
will force signal s to 1 for the duration of the current time period.

See also

noforce command (CR-101), and **change** command (CR-43)

getactivecursortime

The **getactivecursortime** command gets the time of the active cursor in the Wave window.

Returns the time value.

Syntax

```
getactivecursortime
      [-window <wname>]
```

Arguments

-window <wname>
Use this option to specify an instance of the Wave window that is not the default. Otherwise, the default Wave window is used. Optional. Use the **view** command (CR-180) to change the default window.

Examples

```
getactivecursortime
```

Returns:

```
980 ns
```

See also

right | left command (CR-136)

getactivemarkertime

The **getactivemarkertime** command gets the time of the active marker in the List window.

Returns the time value. If -delta is specified, returns time and delta.

Syntax

```
getactivemarkertime
      [-window <wname>] [-delta]
```

Arguments

`-window <wname>`
Use this option to specify an instance of the List window that is not the default. Otherwise, the default List window is used. Optional. Use the **view** command (CR-180) to change the default window.

`-delta`
Also return the delta value. Valid for the List window only. Optional. Default is to return only the time.

Examples

```
getactivemarkertime -delta
```

Returns:

```
980 ns, delta 0
```

See also

right | left command (CR-136), and the **disable_menuitem** command (CR-66)

lecho

The **lecho** command takes one or more Tcl lists as arguments and pretty-prints them to the Transcript window. Returns nothing.

Syntax

```
lecho
      <args> ...
```

Arguments

<args> ...
Any Tcl list created by a VSIM command or user procedure.

Examples

```
lecho [configure wave]
```

Prints the Wave window configuration list to the Transcript window.

log

The **log** command creates a logfile containing simulation data for all HDL items whose names match the provided specifications. Items (VHDL variables, and Verilog nets and registers) that are displayed using the **add list** (CR-22) and **add wave** (CR-33) commands are automatically recorded in the logfile.

If no port mode is specified, the logfile contains data for all items in the selected region whose names match the item name specification.

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

Syntax

```
log
      [-recursive] [-in] [-out] [-inout] [-ports]
      [-internal] [-howmany] <item_name>
```

Arguments

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

`-in`
Specifies that the logfile is to include data for ports of mode IN whose names match the specification. Optional.

`-out`
Specifies that the logfile is to include data for ports of mode OUT whose names match the specification. Optional.

`-inout`
Specifies that the logfile is to include data for ports of mode INOUT whose names match the specification. Optional.

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

`-internal`
Specifies that the logfile is to include data for internal items whose names match the specification. Optional.

`-howmany`
Returns an integer indicating the number of signals found. Optional.

`<item_name>`
Specifies the item name which you want to log. Required. Multiple item names may be specified. Wildcard characters are allowed.

Examples

```
log -r /*
```
Logs all items in the design.

```
log -out *
```
Logs all output ports in the current design unit.

See also

add list command (CR-22), **add wave** command (CR-33), **nolog** command (CR-102), and "Wildcard characters" (CR-248)

lshift

The **lshift** command takes a Tcl list as argument and shifts it in-place one place to the left, eliminating the 0th element. The number of shift places may also be specified. Returns nothing.

Syntax

```
lshift
      <list> [<amount>]
```

Arguments

`<list>`
Specifies the Tcl list to target with **lshift**. Required.

`<amount>`
Specifies more than one place to shift. Optional. Default is 1.

Examples

```
proc myfunc args {

    # throws away the first two arguments

    lshift args 2

    ...

}
```

See also

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

lsublist

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

Syntax

```
lsublist
      <list> <pattern>
```

Arguments

`<list>`
Specifies the Tcl list to target with **lsublist**. Required.

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

Examples

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

```
set window_names "structure signals variables process source wave list dataflow"

set t [lsublist $window_names s*]
```

See also

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

macro_option

This command is available for **UNIX only**.

The **macro_option** command controls the speed and delay of macro (DO file) playback, plus the level of debugging feedback. If invoked without any options **macro_option** returns all current settings; returns a specific setting if invoked with an option and no argument.

Syntax

```
macro_option
      [speed fast | demo] | [delay <delay_time>] | [debug <level>]
```

Arguments

`speed fast | demo`
Set the macro playback speed to fast or demo. Optional.

`delay <delay_time>`
Set the delay time in milliseconds; delay is the time between events in demo mode. Optional.

`debug <level>`
Set debug level from 1 to 9; 9 giving the most feedback. Optional.

See also

play command (CR-110), and the **run** command (CR-139)

modelsim

The **modelsim** command starts the Model*Sim* 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 Model*Sim* shortcut
from the Windows Start > Run menu

To use **modelsim** arguments with a shortcut, add them to the target line of the shortcut's properties. (Arguments work on the DOS command line too, of course.)

The simulator may be invoked from either the MODELSIM prompt after the GUI starts or from a DO file called by **modelsim**.

Syntax

```
modelsim
      [-do <macrofile>] [-project <project file>]
```

Arguments

`-do <macrofile>`
Specifies the DO file to execute when **modelsim** is invoked. Optional.

Note: In addition to the macro called by this argument, if a DO file is specified by the STARTUP variable in *modelsim.ini*, it will be called when the **vsim** command (CR-208) is invoked.

`-project <project file>`
Specifies the *modelsim.ini* file to load for this session. Optional.

See also

vsim command (CR-208), **do** command (CR-67), and "Using a startup file" (B-404)

.main clear

The **.main clear** command clears the Main window (10-158) transcript. The behavior is the same as the Main window **File > Clear Transcript** menu selection.

Syntax

```
.main clear
```

Arguments

None.

next

See "search and next" (CR-141) for information on the **next** command.

noforce

The **noforce** command removes the effect of any active **force** (CR-87) commands on the selected HDL items. The **noforce** command also causes the item's value to be re-evaluated.

Syntax

```
noforce
      <item_name> ...
```

Arguments

`<item_name>`
Specifies the name of a item. Required. Must match the item name used in the **force** command (CR-87). Multiple item names may be specified. Wildcard characters are allowed.

See also

force command (CR-87), and "Wildcard characters" (CR-248)

nolog

The **nolog** command suspends writing of data to the logfile for the specified signals. A flag is written into the logfile for each signal turned off, and the gui displays question marks for the signal until logging (for the signal) is turned back on.

Logging can be turned back on by issuing another **log** command (CR-93) or by doing a **nolog -reset**.

Because use of the **nolog** command adds new information to the logfile, logfiles created when using the **nolog** command cannot be read by older versions of the simulator. If you are using dumplog64.c, you will need to get an updated version.

Syntax

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

Arguments

`-all`
Turns off logging for all signals currently logged.Optional.

`-reset`
Turns logging back on for all signals unlogged. Optional.

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

`-in`
Specifies that the logfile is to exclude data for ports of mode IN whose names match the specification. Optional.

`-out`
Specifies that the logfile is to exclude data for ports of mode OUT whose names match the specification. Optional.

`-inout`
Specifies that the logfile is to exclude data for ports of mode INOUT whose names match the specification. Optional.

`-ports`
Specifies that the scope of the search is to exclude all port. Optional.

`-internal`
Specifies that the logfile is to exclude data for internal items whose names match the specification. Optional.

`-howmany`
Returns an integer indicating the number of signals found. Optional.

`<item_name>`
Specifies the item name which you want to unlog. Required. Multiple item names may be specified. Wildcard characters are allowed.

Examples

```
nolog -r /*
```
Unlogs all items in the design.

```
nolog -out *
```
Unlogs all output ports in the current design unit.

See also

log command (CR-93)

notepad

The **notepad** command is a simple text editor. It may be used to view and edit ascii files or create new files. When a file is specified on the command line, the editor will initially come up in read-only mode. This mode can be changed from the Notepad Edit menu. See "Editing the command line, the current source file, and notepads" (10-167) for a list of editing shortcuts.

Returns nothing.

Syntax

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

Arguments

`<filename>`
Name of the file to be displayed.

`-r | -edit`
Selects the notepad editing mode: -r for read-only, and -edit for edit mode. Optional. Read-only is default.

nowhen

The **nowhen** command deactivates selected **when** (CR-226) commands.

Syntax

```
nowhen
      [<label>]
```

Arguments

`<label>`
Used to identify individual when commands. Optional.

Examples

```
when -label 99 b {echo “b changed”}
…
nowhen 99
```

This **nowhen** command deactivates the **when** (CR-226) command labeled 99.

```
nowhen *
```

This **nowhen** command deactivates all **when** (CR-226) commands.

onbreak

The **onbreak** command is used within a macro; it specifies a command to be executed when running a macro that encounters a breakpoint in the source code. Using the **onbreak** command without arguments will return the current **onbreak** command string. Use an empty string to change the **onbreak** command back to its default behavior (i.e., onbreak ""). In that case, the macro will be interrupted after a breakpoint occurs (after any associated **bp** command (CR-40) string is executed).

onbreak commands can contain macro calls.

Syntax

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

Arguments

`<command>`
Any VSIM command can be used as an argument to **onbreak**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces. It is an error to execute any commands within an **onbreak** command string following a **run** (CR-139), **run -continue**, or **step** (CR-151) command. This restriction applies to any macros or Tcl procedures used in the **onbreak** command string. Optional.

Examples

`onbreak {exa data ; cont}`
Examine the value of the HDL item data when a breakpoint is encountered. Then continue the **run** command (CR-139).

`onbreak {resume}`
Resume execution of the macro file on encountering a breakpoint.

See also

abort command (CR-19), **bd** command (CR-39), **bp** command (CR-40), **do** command (CR-67), **onerror** command (CR-108), **resume** command (CR-135), and the **status** command (CR-150)

onElabError

The **onElabError** command specifies one or more commands to be executed when an error is encountered during elaboration. The command is used by placing it within the *modelsim.tcl* file or a macro. During initial design load **onElabError** may be invoked from within the *modelsim.tcl* file; during a simulation restart **onElabError** may be invoked from a macro.

Use the **onElabError** command without arguments to return to a prompt.

Syntax

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

Arguments

`<command>`
Any VSIM command can be used as an argument to **onElabError**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces. Optional.

See also

do command (CR-67)

onerror

The **onerror** command is used within a macro; it specifies one or more commands to be executed when a running macro encounters an error. Using the **onerror** command without arguments will return the current **onerror** command string. Use an empty string to change the **onerror** command back to its default behavior (i.e., onbreak ""). Use **onerror** with a **resume** command (CR-135) to allow an error message to be printed without halting the execution of the macro file.

Syntax

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

Arguments

<command>

Any VSIM command can be used as an argument to **onerror**. If you want to use more than one command, use a semicolon to separate the commands, or place them on multiple lines. The entire command string must be placed in curly braces. Optional.

See also

do command (CR-67), **onbreak** command (CR-106), **resume** command (CR-135), and the **status** command (CR-150)

pause

The **pause** command placed within a macro interrupts the execution of that macro.

Syntax

```
pause
```

Arguments

None.

Description

When you execute a macro and that macro gets interrupted, the prompt will change to:

```
VSIM (pause)7>
```

This “pause” prompt reminds you that a macro has been interrupted.

When a macro is paused, you may invoke another macro, and if that one gets interrupted, you may even invoke another — up to a nesting level of 50 macros.

If the status of nested macros gets confusing, use the **status** command (CR-150). It will show you which macros are interrupted, at what line number, and show you the interrupted command.

To resume the execution of the macro, use the **resume** command (CR-135). To abort the execution of a macro use the **abort** command (CR-19).

See also

abort command (CR-19), **resume** command (CR-135), and the **run** command (CR-139)

play

This command is available for **UNIX only**.

The **play** command replays a sequence of keyboard and mouse actions, which were previously saved to a file with the **record** command (CR-130). Returns nothing.

Note: Play returns immediately; the playback proceeds in the background. Caution must be used when putting **play** commands in do (macro) files.

Syntax

```
play
      <filename>
```

Arguments

`<filename>`
Specifies the recorded file to replay. Required.

Playback controls

The following Tcl **set** commands control the playback type and speed by setting the **play_macro()** global variables. The commands are invoked from the Model*Sim* command line.

`set play_macro(speed)`
Specify the playback speed: either demo (with the delay specified below), or fast (no delays).

`set play_macro(delay)`
Specifies the delay time in milliseconds. Controls the speed of playback in demo mode.

See also

macro_option command (CR-97), and the **record** command (CR-130)

power add

The **power add** command is used prior to the **power report** command (CR-112). Data produced by these commands can be translated (by a Synopsys utility) to drive the Synopsys power analysis tools. This command specifies the signals or nets to track for power information. Returns nothing.

Syntax

```
power add
      [-in] [-out] [-internal] [-internal] [-r] <signalsOrNets> ...
```

Arguments

`-in`
Select only inputs. Optional.

`-out`
Select only outputs. Optional.

`-internal`
Select only design internal signals or nets. Optional.

`-ports`
Select only design ports. Optional.

`-r`
Do the wildcard search recursively. Optional.

`<signalsOrNets> ...`
Specifies the signal or net to track. Required. Multiple names or wildcards may be used. When using wildcards, switches filter the qualifying signals. The switches select inputs, outputs, internal signals, or ports. If more than one switch is used, the logical OR of the option is performed. This argument must refer to VHDL signals of type bit, std_logic, or std_logic_vector, or to Verilog nets.

See also

power report command (CR-112), and the **power reset** command (CR-113)

See the Synopsys Power documentation for more information.

power report

The **power report** command is used subsequent to the **power add** command (CR-111). Data produced by these commands can be translated (by a Synopsys utility) to drive the Synopsys power analysis tools. This command writes out the power information for the specified signals or nets. The report can be written to a file or to the Transcript window. Returns nothing.

Syntax

```
power report
      [-all] [-noheader] [-file <filename>]
```

Arguments

`-all`
Writes information on all items logged. Optional.

`-noheader`
Suppresses the header to aid in post processing. Optional.

`-file <filename>`
Specifies a filename for the power report. Optional. Default is to write the report to the Transcript window.

Description

The report format for each line is:

```
signal path, toggle count, hazard count, time at a 1, time at a 0, time at an X
```

- toggle count is the number of 0->1 and 1->0 transitions
- hazard count is the number of 0/1->X, and X->0/1 transitions
- times are the times spent at each of the three respective states

You will also need to know the total simulation time.

See also

power add command (CR-111), and the **power reset** command (CR-113)

See the Synopsys Power documentation for more information.

power reset

The **power reset** command selectively resets power information to zero for the signals or nets specified with the **power add** command (CR-111). Returns nothing.

Syntax

```
power reset
      [-in] [-out] [-internal] [-internal] [-r] <signalsOrNets> ...
```

Arguments

`-in`
Reset only inputs. Optional.

`-out`
Reset only outputs. Optional.

`-internal`
Reset only design internal signals or nets. Optional.

`-ports`
Reset only design ports. Optional.

`-r`
Do the wildcard search recursively. Optional.

`<signalsOrNets> ...`
Specifies the signal or net to reset. Required. Multiple names or wildcards may be used.

See also

power add command (CR-111), and the **power report** command (CR-112)

See the Synopsys Power documentation for more information.

printenv

The **printenv** command echoes to the Transcript window the current names and values of all environment variables. If variable names are given as arguments, prints only the names and values of the specified variables. Returns nothing. All results go to the Transcript window.

Syntax

```
printenv
      [<var>...]
```

Arguments

<var>...
Specifies the name(s) of the environment variable to print. Optional.

Examples

```
printenv
```

Prints all environment variable names and their current values (usually a dozen or so):

```
# CC = gcc
# DISPLAY = srl:0.0
...
```

```
printenv USER HOME
```

Prints the specified environment variables:

```
# USER = vince
# HOME = /scratch/srl/vince
```

profile clear

The **profile clear** command is used to clear any data that has been gathered during previous **run** commands. After this command is executed, all profiling data will be reset.

This command has no effect on the current profiling session. The last **profile on** or **profile off** command will still be in effect.

Syntax

```
profile clear
```

Arguments

None

See also

profile interval command (CR-116), **profile off** command (CR-117), **profile on** command (CR-118), **profile option** command (CR-119), **profile report** command (CR-120)

Note: Profiling must be active when this command is invoked. Use the **profile on** command (CR-118) to begin profiling.

profile interval

The **profile interval** command allows you to select the frequency with which the profiler collects samples during a run command.

Syntax

```
profile interval
      [sample-frequency]
```

Arguments

`sample-frequency`
An integer value from 1 to 999 that represents how many milliseconds to wait between each sample collected during a profiled simulation run. Default is 10 ms.

If the sample-frequency is not supplied, the **profile interval** command returns the current sample frequency.

See also

profile clear command (CR-115), **profile off** command (CR-117), **profile on** command (CR-118), **profile option** command (CR-119), **profile report** command (CR-120)

Note: Profiling must be active when this command is invoked. Use the **profile on** command (CR-118) to begin profiling.

profile off

The **profile off** command is used to discontinue runtime profiling.

Syntax

```
profile off
```

Arguments

None

See also

profile clear command (CR-115), **profile interval** command (CR-116), **profile on** command (CR-118), **profile option** command (CR-119), **profile report** command (CR-120)

profile on

The **profile on** command is used to enable runtime analysis of where your simulation is spending its time. After this command is executed, every subsequent **run** command will be profiled.

Syntax

```
profile on
```

Arguments

None

See also

profile clear command (CR-115), **profile interval** command (CR-116), **profile off** command (CR-117), **profile option** command (CR-119), **profile report** command (CR-120)

profile option

The **profile option** command allows various profiling options to be changed.

Syntax

```
profile option
      [collapse_sections] [raw_data]
```

Arguments

`collapse_sections`

By default all profiling data is reported on a per line basis. This option allows you to group the data by section. A section consists of regions of code such as VHDL processes, functions or Verilog *always* blocks.

By specifying this option the current setting is toggled. The default is to not collapse sections.

Note: This option must be specified before the **run** command (CR-139) is executed.

`raw_data`

By default all profiling results are reported on a percentage basis. This option allows reporting based on the raw number of samples that occurred in a line or a section.

By specifying this option the current setting is toggled. The default is to display results as percentages.

See also

profile interval command (CR-116), **profile interval** command (CR-116), **profile off** command (CR-117), **profile on** command (CR-118), **profile report** command (CR-120)

Note: Profiling must be active when this command is invoked. Use the **profile on** command (CR-118) to begin profiling.

profile report

The **profile report** command is used to produce a textual output of the profiling statistics that have been gathered up to this point. (The **View** > **Other** > **Hierarchical Profile** and **View** > **Other** > **Ranked Profile** menu picks allow you to view this data more interactively.)

Syntax

```
profile report
      [ -hierarchical | -ranked ] [ -file <filename> ] [ -cutoff
      <percentage> ]
```

Arguments

`-hierarchical`
Report a hierarchical listing (Default). Optional.

`-ranked`
Report a ranked listing. Optional.

`-file <filename>`
Specifies a file name for the report. Optional. Default is to write report to the Transcript Window.

`-cutoff <percentage>`
Filter entries in the report that had less than <percentage> of time spent in them. Optional. Default is report all entries (i.e. 0%).

See also

profile interval command (CR-116), **profile interval** command (CR-116), **profile off** command (CR-117), **profile on** command (CR-118), **profile option** command (CR-119)

Note: Profiling must be active when this command is invoked. Use the **profile on** command (CR-118) to begin profiling.

project

The **project** command is used to perform common operations on new projects. The command is to be used outside of a simulation session.

Syntax

```
project
      [history] [compile] [close] [env] [save] [copy <src_project>
      <dest_dir> <proj_name>] [new <home_dir> <proj_name>] [delete
      <project> [-force]] [open <project>]
```

Arguments

`history`
Lists a history of manipulated projects.

`new <home_dir> <proj_name>`
Creates a new project under a specified home directory with a specified name.

`copy <src_project> <dest_dir> <proj_name>`
Copies an existing project to a destination directory with a specified name.

`open <project>`
Opens a specified project file making it the current project. Changes the current working directory to the project's directory.

`env`
Returns the current project file.

`compile`
Compile the current project. This is done by taking the project's list of work libraries (Work_Libs variable in project section) and executing the associated build script for each library (<libname>_script variable in the project section).

`delete <project> [-force]`
Deletes a specified project.

`close`
Close the current project.

`save`
Update the current project's .mpf file with the current **vsim** settings.

Examples

The following commands make a copy of /user/georges/design/test3 project at /user/georges/design/test4.

```
vsim> project history
# 0 /user/george/design/test3/test3.mpf
# 1 /user/george/design/test2/test2.mpf
# 2 /user/george/design/test1/test1.mpf
# 3 /home/prod/release/5.3/modeltech/examples/projects/mixed/mixed.mpf
# 4 /home/prod/release/5.3/modeltech/examples/projects/verilog/verilog.mpf
# 5 /home/prod/release/5.3/modeltech/examples/projects/vhdl/vhdl.mpf
vsim> project copy !0 /user/georges/design test4
```

The following command makes /user/george/design/test3 the current project and changes the current working directory to /user/george/design/test3.

```
vsim> project open /user/george/design/test3/test3.mpf
```

The folowing command causes execution of current project library build scripts.

```
 vsim> project compile
```

The following command writes the current vsim tool settings to the current project file.

```
vsim> project save
```

The following commands delete /user/georges/test2/test2.mpf

```
vsim> project history
# 0 /user/george/design/test4/test4.mpf
# 1 /user/george/design/test3/test3.mpf
# 2 /user/george/design/test2/test2.mpf
# 3 /user/george/design/test1/test1.mpf
# 4 /home/prod/release/5.3/modeltech/examples/projects/mixed/mixed.mpf
# 5 /home/prod/release/5.3/modeltech/examples/projects/verilog/verilog.mpf
# 6 /home/prod/release/5.3/modeltech/examples/projects/vhdl/vhdl.mpf
vsim> project delete !2
```

property list

The **property list** command changes one or more properties of the specified signal, net or register in the List window (10-174). The properties correspond to those than can be specified using the List window **Prop > Display Props** menu selection. At least one argument must be used.

Syntax

```
property list
      [-window <wname>>] [-label <label>] [-radix <radix>]
      [-trigger <setting>] [-width <number>] <pattern>
```

Arguments

`-window <wname>`
Used to specify a particular List window when multiple instances of the window exist (i.e., list2). Optional. If no window is specified the default window is used; the default window is determined by the most recent invocation of the **view** command (CR-180).

`-label <label>`
Specifies the label to appear at the top of the List window column. Optional.

`-radix <radix>`
The listed value <radix> can be specified as: Symbolic, Bin, Oct, Dec or Hex. Optional.

`-trigger <setting>`
Valid settings are 0 or 1. Setting trigger to 1 will enable the list window to be triggered by changes on this signal. Optional.

`-width <number>`
Valid numbers are 1 through 256. Specifies the desired column width for the listed <pattern>. Optional.

`<pattern>`
Specifies a name or wildcard pattern to match the full path names of the signals, nets or registers for which you are defining the property change. Required.

To change the time or delta column widths, use these patterns:

```
TIME or DELTA
```

property wave

The **property wave** command changes one or more properties of the specified signal, net or register in the Wave window (10-212). The properties correspond to those than can be specified using the Wave window **Prop > Display Props** menu selection. At least one argument must be used.

Syntax

```
property wave
     [-window <wname>] [-color <color>] [-format <format>] [-height
     <number>] [-offset <number>] [-radix <radix>] [-scale <float>]
     <pattern>
```

Arguments

`-window <wname>`
Used to specify a particular Wave window when multiple instances of the window exist (i.e., wave2). Optional. If no window is specified the default window is used; the default window is determined by the most recent invocation of the **view** command (CR-180).

`-color <color>`
Specifies any valid system color name. Optional.

`-format <format>`
The waveform <format> can be expressed as:

`analog`
Displays a waveform whose height and position is determined by the -scale and -offset values (shown below). Optional.

`literal`
Displays the waveform as a box containing the item value (if the value fits the space available). Optional.

`logic`
Displays values as 0, 1, X, or Z. Optional.

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

`-offset <number>`
Specifies the waveform position offset in pixels. Valid only when **-format** is specified as analog. Optional.

`-radix <radix>`
The <radix> can be expressed as: Symbolic, Bin, Oct, Dec or Hex. Choosing symbolic means that item values are not translated. Optional.

`-scale <float>`
Specifies the waveform scale relative to the unscaled size value of 1. Valid only when **-format** is specified as analog. Optional.

`<pattern>`
Specifies a name or wildcard pattern to match the full path names of the signals, nets or registers for which you are defining the property change. Required.

pwd

The Tcl **pwd** command displays the current directory path in the Main transcript window.

Returns nothing.

Syntax

```
pwd
```

Arguments

None.

quietly

The **quietly** command turns off transcript echoing for the specified command.

Syntax

```
quietly
      <command>
```

Arguments

`<command>`
Specifies the command for which to disable transcript echoing. Required. Any results normally echoed by the specified command will not be written to the Main window transcript. To disable echoing for all commands use the **transcript** command (CR-158) with the **-quietly** option.

See also

transcript command (CR-158)

quit

The **quit** command exits the simulator.

Syntax

```
quit
      [-f or -force] [-sim]
```

Arguments

`-f or -force`
Quits without asking for confirmation. Optional; if this argument is omitted, VSIM asks you for confirmation before exiting. (The -f and -force arguments are equivalent.)

`-sim`
Unloads the current design in the simulator without exiting Mod*elSim*. All files opened by the simulation will be closed including the logfile (*vsim.wav*).

Note: If you want to stop the simulation using a **when** command (CR-226), you must use a **stop** command (CR-152) within your when statement. DO NOT use an **exit** command (CR-84) or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated.

radix

The **radix** command specifies the default radix to be used. The command can be used at any time. The specified radix is used for all commands (**force** (CR-87), **examine** (CR-81), **change** (CR-43), etc.) as well as for displayed values in the Signals, Variables, Dataflow, List, and Wave windows.

Syntax

```
radix
      [-symbolic | -binary | -octal | -decimal | -hexadecimal |
      -unsigned | -ascii]
```

Arguments

Entries may be truncated to any length, for example, -symbolic could be expressed as -s or -sy, etc. Optional.

Also, -signed may be used as an alias for -decimal. The -unsigned radix will display as unsigned decimal. The -ascii radix will display a Verilog item as a string equivalent using 8 bit character encoding.

If no arguments are used, the command returns the current default radix.

record

This command is available for **UNIX only**.

The **record** command starts recording a replayable trace of all keyboard and mouse actions. Record and play operations may also be run from the macro-helper menu item of the macro menu. Returns nothing.

Syntax

```
record
      [<filename>]
```

Arguments

`<filename>`
Specifies the file for the saved recording. If <filename> is not specified, the recording terminates.

See also

macro_option command (CR-97), and the **play** command (CR-110)

report

The **report** command displays the value of all simulator control variables, or the value of any simulator state variables relevant to the current simulation.

Syntax

```
report
      simulator control | simulator state
```

Arguments

`simulator control`
Displays the current values for all simulator control variables.

`simulator state`
Displays the simulator state variables relevant to the current simulation.

Examples

`report simulator control`
Displays all simulator control variables.

```
# UserTimeUnit = ns
# RunLength = 100
# IterationLimit = 5000
# BreakOnAssertion = 3
# DefaultForceKind = default
# IgnoreNote = 0
# IgnoreWarning = 0
# IgnoreError = 0
# IgnoreFailure = 0
# CheckpointCompressMode = 1
# NumericStdNoWarnings = 0
# StdArithNoWarnings = 0
# PathSeparator = /
# DefaultRadix = symbolic
# DelayFileOpen = 0
# WaveSignalNameWidth = 0
# ListDefaultShortName = 1
# ListDefaultIsTrigger = 1
```

`report simulator state`

Displays all simulator state variables. Only the variables that relate to the design being simulated are displayed:

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

Viewing preference variables

Preference variables have more to do with the way things look (but not entirely) rather then controlling the simulator. You can view preference variables from the Preferences dialog box. Select the Main menu: Options > Edit Preferences, or see "Setting variables with the GUI" (B-413).

See also

"Preference variables located in INI and MPF files" (B-394), and "Preference variables located in TCL files" (B-406)

restart

The **restart** command reloads the design elements and resets the simulation time to zero. Only design elements that have changed are reloaded.

Syntax

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

Arguments

`-force`
Specifies that the simulation will be restarted without requiring confirmation in a popup window. Optional (unless being used in a macro file).

`-nobreakpoint`
Specifies that all breakpoints will be removed when the simulation is restarted. Optional. The default is for all breakpoints to be reinstalled after the simulation is restarted.

`-nolist`
Specifies that the current List window environment will **not** be maintained after the simulation is restarted. Optional. The default is for all currently listed HDL items and their formats to be maintained.

`-nolog`
Specifies that the current logging environment will **not** be maintained after the simulation is restarted. Optional. The default is for all currently logged items to continue to be logged.

`-nowave`
Specifies that the current Wave window environment will **not** be maintained after the simulation is restarted. Optional. The default is for all items displayed in the Wave window to remain in the window with the same format.

See also

checkpoint (CR-53)**,** and "How to use checkpoint/restore" (E-438).

restore

The **restore** command restores the state of a simulation that was saved with a **checkpoint** command (CR-53) during the current invocation of VSIM (called a "warm restore").

The items restored are: simulation kernel state, *vsim.wav* file, HDL items listed in the List and Wave windows, file pointer positions for files opened under VHDL and under Verilog $fopen, and the saved state of foreign architectures.

If you want to **restore** while running VSIM, use this command. If you want to start up VSIM and restore a previously-saved checkpoint, use the **-restore** switch with the **vsim** command (CR-208), this we call a "cold restore".

Note: Checkpoint/restore allows a cold restore, followed by simulation activity, followed by a warm restore back to the original cold-restore checkpoint file. Warm restores to checkpoint files that were not created in the current run are not allowed except for this special case of an original cold restore file.

Syntax

```
restore
      [-nocompress] <filename>
```

Arguments

`-nocompress`
Specifies that the checkpoint file was not compressed when saved. Optional.

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

See also

checkpoint command (CR-53), and "The difference between checkpoint/restore and restarting" (E-439).

resume

The **resume** command is used to resume execution of a macro file after a **pause** command (CR-109), or a breakpoint. It may be input manually or placed in an **onbreak** (CR-106) command string. (Placing a **resume** command in a **bp** (CR-40) command string does not have this effect.) The **resume** command can also be used in an **onerror** (CR-108) command string to allow an error message to be printed without halting the execution of the macro file.

Syntax

```
resume
```

Arguments

None.

See also

abort command (CR-19), **pause** command (CR-109), and the **do** command (CR-67)

right | left

The **right | left** command searches right (next) or left (previous) for signal transitions or values in the specified Wave window. It executes the search on signals currently selected in the window, starting at the time of the active cursor. A condition to search for may also be identified by an expression using the **-expr** command option. The active cursor moves to the found location.

Use this command to move to consecutive transitions or to find the time at which a waveform takes on a particular value, or an expression of multiple signals evaluates to true. Use the mouse to select the desired waveform and click on the desired starting location using the left mouse button. Then issue the **right | left** command. (The **seetime** command (CR-146) can initially position the cursor from the command line, if desired.)

Returns: <number_found> <new_time> <new_delta>

Syntax

```
right | left
      [-expr {<expression>}] [-falling] [-noglitch] [-rising]
      [-value <sig_value>] [-window <wname>] [<n>]
```

Arguments

`-expr {<expression>}`
The waveform display will be searched until the expression evaluates to a boolean true condition. Optional. The expression may involve more than one signal, but is limited to signals that have been logged in the referenced Wave window. A signal may be specified either by its full path or by the shortcut label displayed in the Wave window.

See "GUI_expression_format" (CR-250) for the format of the expression. The expression must be placed within curly braces.

`-falling`
Searches for a falling edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-noglitch`
Looks at signal values only on the last delta of a time step. For use with -value option only. Optional.

`-rising`
Searches for a rising edge on the specified signal if that signal is a scalar signal. If it is not a scalar signal, the option will be ignored. Optional.

`-value <sig_value>`
Specify a value of the signal to match. Must be specified in the same radix that the selected waveform is displayed. Case is ignored, but otherwise must be an exact string match -- don't-care bits are not yet implemented. Only one signal may be selected, but that signal may be an array. Optional.

`-window <wname>`
Use this option to specify an instance of the Wave window that is not the default. Optional. Otherwise, the default Wave window is used. Use the **view** command (CR-180) to change the default window.

`<n>`
Specifies to find the nth match. If less than n are found, the number found is returned with a warning message, and the cursor is positioned at the last match. Optional. The default is 1.

Examples

`right -noglitch -value FF23 2`
Finds the second time to the right at which the selected vector transitions to FF23, ignoring glitches.

`left`
Goes to the previous transition on the selected signal.

The following examples illustrate search expressions that use a variety of signal attributes, paths, array constants, and time variables. Such expressions follow the "GUI_expression_format" (CR-250) and can be built with the aid of the "The GUI Expression Builder" (10-272).

```
right -expr {clk'rising && (mystate == reading) && (/top/u3/addr ==
    32'habcd1234)}
```
Searches right for an expression that evaluates to a boolean 1 when signal clk just changed from low to high and signal mystate is the enumeration reading and signal /top/u3/addr is equal to the specified 32-bit hex constant; otherwise is 0.

```
right -expr {(/top/u3/addr and 32'hff000000) == 32'hac000000}
```
Searches right for an expression that evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal /top/u3/adder equals hex ac.

```
right -expr {((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode == writing)}
```

Searches right for an expression that evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, and clock just changed from low to high and signal mode is enumeration writing.

Note: Wave window keyboard shortcuts (10-237) are also available for next and previous edge searches. Tab searches right (next) and shift-tab searches left (previous).

See also

"GUI_expression_format" (CR-250), **view** command (CR-180), and the **seetime** command (CR-146)

run

The **run** command advances the simulation by the specified number of timesteps.

Syntax

```
run
      [<timesteps> [<time_units>]]| -all | -continue | -next | -step |
      -stepover]
```

Arguments

`<timesteps>[<time_units>]`
Specifies the number of timesteps for the simulation to run. The number may be fractional. Optional. In addition, optional <time_units> may be specified as:

fs, ps, ns, us, ms, or **sec**

The default <timesteps> and <time_units> specifications can be changed during a VSIM session from the **Options > Simulation** menu option in the Main window (see "Setting default simulation options" (10-260)). Time steps and time units may also be set with the RunLength (B-400) and UserTimeUnit (B-401) variables in the *modelsim.ini* file.

`-all`
Causes the simulator to run until there are no events scheduled. Optional.

`-continue`
Continues the last simulation run after a **step** (CR-151) command, **step -over** command or a breakpoint. A **run -continue** command may be input manually or used as the last command in a **bp** (CR-40) command string. Optional.

`-next`
Causes the simulator to run to the next event time. Optional.

`-step`
Steps the simulator to the next HDL statement. Optional.

`-stepover`
Specifies that VHDL procedures, functions and Verilog tasks are to be executed but treated as simple statements instead of entered and traced line by line. Optional.

Examples

`run 1000`
Advances the simulator 1000 timesteps.

`run 10.4 ms`
Advances the simulator the appropriate number of timesteps corresponding to 10.4 milliseconds.

`run @8000`
Advances the simulator to timestep 8000.

See also

step command (CR-151)

search and next

The **search** and **next** commands search the specified window for one or more items matching the specified pattern(s). The search starts at the item currently selected, if any; otherwise starts at the window top. Default action is to search downward until the first match, then move the selection to the item found, and return the index of the item found. The search can be continued using the **next** command.

Returns the index of a single match, or list of matching indices. Returns nothing if no matches are found.

Syntax

```
search
      <win_type> [-window <wname>] <pattern> [-reverse] [-all]
      [-field <n>] [-toggle] [-forwards] [-backwards] [-exact] [-regexp]
      [-nocase] [-count <n>]
next
      <win_type> [-window <wname>]
```

Arguments

`<win_type>`
Specifies structure, signals, process, variables, wave, list, source, or a unique abbreviation thereof. Required.

`-window <wname>`
Use this option to specify an instance of the window that is not the default. Optional. Otherwise, the default window is used. Use the **view** command (CR-180) to change the default window.

`<pattern>`
String or glob-style wild-card pattern. Required.

Arguments, for all EXCEPT the Source window

`-reverse`
Search in the reverse direction. Optional. Default is forward.

`-all`
Find all matches and return a list of the indices of all items that match. Optional.

`-field <n>`
Selects different fields to test, depending on the window type:

Window	n=1	n=2	n=3	default
structure	instance	ent/mod	[arch]	instance
signals	name	-	cur. value	name
process	status	label	fullpath	fullpath
variables	name	-	cur. value	name
wave	name	-	cur. value	name
list	label	fullname	-	label

Default behavior for the List window is to attempt to match the label and if that fails, try to match the full signal name.

`-toggle`
Adds signals found to the selection. Does not do an initial clear selection. Optional. Otherwise deselects all and selects only one item.

Arguments, Source window only

`-forwards`
Search in the forward direction. This is the default.

`-backwards`
Search in the reverse direction. Optional. Default is forwards.

`-exact`
Search for an exact match.

`-regexp`
Use the pattern as a Tcl regular expression. Optional.

`-nocase`
Ignore case. Optional. Default is to use case.

`-count <n>`
Search for the nth match. Optional. Default is to search for the first match.

Description

With the **-all** option, the entire window is searched, the last item matching the pattern is selected, and a Tcl list of all corresponding indices is returned.

With the **-toggle** option, items found are selected in addition to the current selection.

For the List window, the search is done on the names of the items listed, that is, across the header. To search for values of signals in the List window, use the **down | up** command (CR-70). Likewise, in the Wave window, the search is done on signal names. To search for signal values in the Wave window, use the **right | left** command (CR-136). The **Edit > Search** menu selection may also be used in both windows.

See also

view command (CR-180)

searchLog

The **searchLog** command searches one or more of the currently open logfiles for a specified condition. It can be used to search for rising or falling edges, for signals equal to a specified value, or for when a generalized expression becomes true.

Syntax

```
searchLog
      <options> <startTime> <pattern>
```

If at least one match is found, it returns the time (and optionally delta) at which the last match occurred and the number of matches found, in a Tcl list:

```
{{<time>} <matchCount>}
```

where **<time>** is in the format **<number>** **<unit>**. If the **-deltas** option is specified, the delta of the last match is also returned:

```
{{<time>} <delta> <matchCount>}
```

If no matches are found, a TCL_ERROR is returned. If one or more matches are found, but less than the number requested, it is not considered an error condition, and the time of the farthest match is returned, with the count of the matches found.

Arguments

`-rising|falling|anyedge`
Specifies an edge to look for on a scalar signal. Optional. This option is ignored for compound signals.

`-reverse`
Specifies to search backwards in time from <startTime>. Optional.

`-deltas`
Indicates to test for match on simulation delta cycles. Otherwise, matches are only tested for at the end of each simulation time step. Optional.

`-expr{<expr>}`
Specifies a general expression of signal values and simulation time. Optional. **searchLog** will search until the expression evaluates to true. The expression must have a boolean result type.

`-value<string><radix>`
Specifies to search until a single scalar or compound signal takes on this value. <**radix**> is the radix of the value string. Optional.

`-count<num>`
Specifies to search for the <num>-th occurrence of the match condition, where **<num>** is a positive integer. Optional.

`-startDelta<num>`
Indicates a simulation delta cycle on which to start. Optional.

`-env<path>`
Provides a design region to look for the signal names. Optional.

`<startTime>`
Specifies the simulation time at which to start the search. Required. The time may be specified as an integer number of simulation units, or as **{<num> <timeUnit>}**, where **<num>** can be integer or with a decimal point, and **<timeUnit>** is one of the standard VHDL time units (fs, ps, ns, us, ms, sec, min, hr).

`<pattern>`
The **<pattern>** argument specifies one or more signal names or wildcard patterns of signal names to search on. Required (unless the -expr option is used).

See also

virtual signal command (CR-193), **virtual function** command (CR-184), **virtual region** command (CR-190), **virtual delete|describe|define** command (CR-182), **virtual expand** command (CR-183), **vmap** command (CR-207), **virtual log|nolog** command (CR-189), **virtual save** command (CR-191), **virtual show|count** command (CR-192)

seetime

The **seetime** command scrolls the List or Wave window to make the specified time visible. For the List window, a delta can be optionally specified as well.

Returns: nothing

Syntax

```
seetime
      list|wave [-window <wname>] [-select] [-delta <num>] <time>
```

Arguments

`list|wave`
Specifies the target window type. Required.

`-window <wname>`
Use this option to specify an instance of the Wave or List window that is not the default. Optional. Otherwise, the default wave or List window is used. Use the **view** command (CR-180) to change the default window.

`-select`
Also move the active cursor or marker to the specified time (and optionally, delta). Optional. Otherwise, the window is only scrolled.

`-delta <num>`
For the List window when deltas are not collapsed, this option specifies a delta. Optional. Otherwise, delta 0 is selected.

`<time>`
Specifies the time to be made visible. Required.

shift

The **shift** command shifts macro parameter values down one place, so that the value of parameter $2 is assigned to parameter $1, the value of parameter $3 is assigned to $2, etc. The previous value of $1 is discarded.

The **shift** command and macro parameters are used in macro files. If a macro file requires more than nine parameters, they can accessed using the **shift** command.

To determine the current number of macro parameters, use the argc (B-420).

Syntax

```
shift
```

Arguments

None.

Description

For a macro file containing nine macro parameters defined as $1 to $9, one **shift** command shifts all parameter values one place to the left. If more than nine parameters are named, the value of the tenth parameter becomes the value of $9 and can be accessed from within the macro file.

See also

do command (CR-67)

show

The **show** command lists HDL items and subregions visible from the current environment. The items listed include:

- **VHDL**
  signals, and instances
- **Verilog**
  nets, registers, tasks, functions, instances and memories

The **show** command returns its results as a formatted Tcl string; to eliminate formatting, use the **Show** command.

Syntax

```
show
      [-all] [<pathname>]
```

Arguments

`-all`
Display all names at and below the specified path recursively. Optional.

`<pathname>`
Specifies the pathname of the environment for which you want the items and subregions to be listed. Optional; if omitted, the current environment is assumed.

Examples

`show`
List the names of all the items and subregion environments visible in the current environment.

`show /uut`
List the names of all the items and subregions visible in the environment named /uut.

`show sub_region`
List the names of all the items and subregions visible in the environment named sub_region which is directly visible in the current environment.

See also

enable_menu command (CR-78), and the **find** command (CR-85)

splitio

The **splitio** command operates on a VHDL inout or out port to create a new signal having the same name as the port suffixed with "__o". The new signal mirrors the output driving contribution of the port.

Syntax

```
splitio
      [-outalso | -outonly] [-r] <signal_name> ...
```

Arguments

`-outalso`
Allows **splitio** to work on out ports as well as inout ports. Optional.

`-outonly`
Allows **splitio** to work *only* on out ports. Optional.

`-r`
Specifies that the port selection occurs recursively into subregions. Optional; if omitted, included ports are limited to the current region.

`<signal_name>...`
Specifies the VHDL port. Operates only on inout ports by default; out ports may be specified with the options above. Separate multiple port names with spaces. Required.

Examples

The **splitio** command operates on inout or out ports and silently ignores any other signals specified. The new signals created may be specified in any **vsim** (CR-208) commands that operate on signals. These signals appear to be out ports to the signal selection options on **vsim** commands. For example,

`splitio /data`
creates signal data_o if data is an inout port.

`add list data`
will list both data and data_o.

status

The **status** command lists all current interrupted macros. The listing shows the name of the interrupted macro, the line number at which it was interrupted, and prints the command itself. It also displays any **onbreak** (CR-106) or **onerror** (CR-108) commands that have been defined for each interrupted macro.

Syntax

```
status
```

Arguments

None.

Examples

The transcript below contains examples of **resume** (CR-135), and **status** commands.

```
VSIM (pause) 4> 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 (pause) 5> resume
# Resuming execution of macro resume_test.do at line 4
```

See also

abort command (CR-19), **do** command (CR-67), **pause** command (CR-109), and the **resume** command (CR-135)

step

The **step** command steps to the next HDL statement. Current values of local variables may be observed at this time using the variables window. VHDL procedures, functions and Verilog tasks can optionally be skipped over. When a wait statement or end of process is encountered, time advances to the next scheduled activity. The Process and Source windows will then be updated to reflect the next activity.

Syntax

```
step
      [-over] [<n>]
```

Arguments

`-over`
Specifies that VHDL procedures, functions and Verilog tasks are to be executed but treated as simple statements instead of entered and traced line by line. Optional.

`<n>`
Any integer. Optional. Will execute ‘n’ steps before returning.

See also

run command (CR-139)

stop

The **stop** command is used with the **when** command (CR-226) to stop simulation in batch files. The **stop** command has the same effect as hitting a breakpoint. The **stop** command may be placed anywhere within the body of the when command.

Syntax

```
stop
```

Arguments

None.

Note: Use the **run** command (CR-139) with the **-continue** option to continue the simulation run, or the **resume** command (CR-135) to continue macro execution. If you want macro execution to resume automatically, put the **resume** command at the top of your macro file:

```
onbreak {resume}
```

Note: If you want to stop the simulation using a **when** command (CR-226), you must use a **stop** command within your when statement. DO NOT use an **exit** command (CR-84) or a **quit** command (CR-128). The **stop** command acts like a breakpoint at the time it is evaluated.

See also

bp command (CR-40)

tb

The **tb** (traceback) command displays a stack trace for the current process in the Main 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.

toggle add

The **toggle add** command enables collection of toggle statistics for the specified nodes. The allowed nodes are Verilog nets and VHDL signals of type bit, bit_vector, std_logic, and std_logic_vector (other types are silently ignored).

Syntax

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

Arguments

`-r`
Specifies that toggle statistic collection is enabled recursively into subregions. Optional; if omitted, toggle statistic collection is limited to the current region.

`-in`
Enables toggle statistic collection on nodes of mode IN. Optional.

`-out`
Enables toggle statistic collection on nodes of mode OUT. Optional.

`-inout`
Enables toggle statistic collection on nodes of mode INOUT. Optional.

`-internal`
Enables toggle statistic collection on internal items. Optional.

`-ports`
Enables toggle statistic collection on nodes of modes IN, OUT, or INOUT. Optional.

`<node_name>`
Enables toggle statistic collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

"Toggle checking" (E-447), **toggle reset** command (CR-155), and the **toggle report** command (CR-156)

toggle reset

The **toggle reset** command resets the toggle counts to zero for the specified nodes.

Syntax

```
toggle reset
      [-all] [-r] [-in] [-out] [-inout] [-internal]
      [-ports] <node_name>
```

Arguments

`-all`
Resets toggle statistic collection for all nodes that have toggle checking enabled. Optional.

`-r`
Specifies that toggle statistic collection is reset recursively into subregions. Optional; if omitted, the reset is limited to the current region.

`-in`
Resets toggle statistic collection on nodes of mode IN. Optional.

`-out`
Resets toggle statistic collection on nodes of mode OUT. Optional.

`-inout`
Resets toggle statistic collection on nodes of mode INOUT. Optional.

`-internal`
Resets toggle statistic collection on internal items. Optional.

`-ports`
Resets toggle statistic collection on nodes of modes IN, OUT, or INOUT. Optional.

`<node_name>`
Resets toggle statistic collection for the named node(s). Required. Multiple names and wildcards are accepted.

See also

"Toggle checking" (E-447), **toggle add** command (CR-154), and the **toggle report** command (CR-156)

toggle report

By default the **toggle report** command displays to the screen a list of all nodes that have not transitioned to both 0 and 1 at least once. Also displayed is a summary of the number of nodes checked, the number that toggled, the number that didn't toggle, and a percentage that toggled.

Syntax

```
toggle report
      [-file <filename>] [-summary] [-all]
```

Arguments

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

`-summary`
Selects only the summary portion of the report. Optional.

`-all`
Lists all nodes checked along with their individual transition to 0 and 1 counts. Optional.

See also

"Toggle checking" (E-447), **toggle add** command (CR-154), and the **toggle reset** command (CR-155)

transcribe

The **transcribe** command displays a command in the Main window, then executes the command. **Transcribe** directs commands to the Main window transcript from an external event such as a menu pick or button selection, it may not be used from the command line or a macro. The **add button** (CR-20) and **add_menuitem** (CR-30) commands can utilize **transcribe**. Returns nothing.

Syntax

```
transcribe
      <command>
```

Arguments

`<command>`
Specifies the command to execute. Required.

Examples

`add button pwd {transcribe pwd} NoDisable`
Creates a button labeled "pwd" that invokes **transcribe** with the **pwd** Tcl command, and echoes the command and its results to the Main window. The button remains active during a run.

See also

add button command (CR-20), and **add_menu** command (CR-26)

transcript

The **transcript** command controls echoing of commands executed in a macro file; also works at top level in batch mode. If no option is specified, the current setting is reported.

Syntax

```
transcript
      [on | off | -q | quietly]
```

Arguments

`on`
Specifies that commands in a macro file will be echoed to the Transcript window as they are executed. Optional.

`off`
Specifies that commands in a macro file will not be echoed to the Transcript window as they are executed. Optional.

`-q`
Returns "0" if transcripting is turned off or "1" if transcripting is turned on. Useful in a Tcl conditional expression. Optional.

`quietly`
Turns off the transcript echo for all commands. To turn off echoing for individual commands see the **quietly** command (CR-127). Optional.

Examples

`transcript on`
Commands within a macro file will be echoed to the Transcript window as they are executed.

`transcript`
If issued immediately after the previous example, the message:

```
Macro transcripting is turned on.
```

would appear in the Transcript window.

See also

echo command (CR-75), and the **transcribe** command (CR-157)

tssi2mti

The **tssi2mti** command is used to convert a vector file in Summit Design Standard Events Format into a sequence of VSIM **force** (CR-87) and **run** (CR-139) commands. The stimulus is written to the standard output.

The source code for **tssi2mti** is provided in the file *tssi2mti.c* in the *examples* directory.

Syntax

```
tssi2mti
      <signal_definition_file> [<sef_vector_file>]
```

Arguments

`<signal_definition_file>`
Specifies the name of the Summit Design signal definition file describing the format and content of the vectors. Required.

`<sef_vector_file>`
Specifies the name of the file containing vectors to be converted. If none is specified, standard input is used. Optional.

Examples

```
tssi2mti trigger.def trigger.sef > trigger.do
```
The command will produce a do file named *trigger.do* from the signal definition file *trigger.def* and the vector file *trigger.sef*.

```
tssi2mti trigger.def < trigger.sef > trigger.do
```
This example is exactly the same as the previous one, but uses the standard input instead.

See also

force command (CR-87), **run** command (CR-139), and the **write tssi** command (CR-236)

vcd add

The **vcd add** command adds the specified items to the VCD file. The allowed items are Verilog nets and variables and VHDL signals of type bit, bit_vector, std_logic, and std_logic_vector (other types are silently ignored). All **vcd add** commands must be executed at the same simulation time. The specified items are added to the VCD header and their subsequent value changes are recorded in the VCD file.

Related Verilog task: $dumpvars

Syntax

```
vcd add
      [-r] [-in] [-out] [-inout] [-internal] [-ports] <item_name>...
```

Arguments

`-r`
Specifies that signal and port selection occurs recursively into subregions. Optional; if omitted, included signals and ports are limited to the current region.

`-in`
Includes ports of mode IN. Optional.

`-out`
Includes ports of mode OUT. Optional.

`-inout`
Includes ports of mode INOUT. Optional.

`-internal`
Includes internal items. Optional.

`-ports`
Includes all ports of modes IN, OUT, or INOUT. Optional.

`<item_name>`
Specifies the Verilog or VHDL item to add to the VCD file. Required. Multiple items may be specified by separating names with spaces. Wildcards are accepted.

See also

See User's Manual *Chapter 13 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.

vcd checkpoint

The **vcd checkpoint** command dumps the current values of all VCD variables to the VCD file. While simulating, only value changes are dumped.

Related Verilog task: $dumpall

Syntax

```
vcd checkpoint
```

Arguments

None.

See also

See *Chapter 13 - Value Change Dump (VCD) Files* for more information on VCD files.

vcd comment

The **vcd comment** command inserts the specified comment in the VCD file.

Syntax

```
vcd comment
      <comment string>
```

Arguments

`<comment string>`
Comment to be included in the VCD file. Required. Must be quoted by double quotation marks or curly brackets.

See also

See *Chapter 13 - Value Change Dump (VCD) Files* for more information on VCD files.

vcd file

The **vcd file** command specifies the filename and state mapping for the VCD file created by a **vcd add** command (CR-160). The **vcd file** command is optional. If used, it must be issued before any **vcd add** commands.

Related Verilog task: $dumpfile

Syntax

```
vcd file
      [<filename>] [-nomap] [-map <mapping pairs>] [-direction]
      [-dumpports]
```

Arguments

`<filename>`
Specifies the name of the VCD file that is created (the default is *dump.vcd*). Optional.

See also

See *Chapter 13 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.

`-nomap`
Affects only VHDL signals of type std_logic. Optional. It specifies that the values recorded in the VCD file shall use the std_logic enumeration characters of UX01ZWLH-. This option results in a non-standard VCD file because VCD values are limited to the four state character set of x01z. By default, the std_logic characters are mapped as follow

VHDL	VCD	VHDL	VCD
U	x	W	x
X	x	L	0
0	0	H	1
1	1	-	x
Z	z		

`-map <mapping pairs>`

Affects only VHDL signals of type std_logic. Optional. It allows you to override the default mappings. The mapping is specified as a list of character pairs. The first character in a pair must be one of the std_logic characters UX01ZWLH- and the second character is the character you wish to be recorded in the VCD file. For example, to map L and H to z:

```
vcd file -map "L z H z"
```

Note that the quotes in the example above are a Tcl convention for command strings that include spaces.

`-direction`

Affects only VHDL ports. Optional. It specifies that the variable type recorded in the VCD header for VHDL ports shall be one of the following:

in, out, inout, internal, ports (includes in, out, and inout); the default is all ports

This option results in a non-standard VCD file, but is necessary if the VCD file is to be used to stimulate a VHDL design with the **vsim** command (CR-208) with the **-vcdread** option.

Note that the port type is specified with options to the **vcd add** command (CR-160) when the VCD file is created.

`-dumpports`

Capture detailed port driver data for Verilog ports and VHDL std_logic ports. Optional. This option only works on ports, and subsequent **vcd add** (CR-160) will only accept qualifying ports (silently ignoring all other specified items).

See also

See *Chapter 13 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.

vcd flush

The **vcd flush** command flushes the contents of the VCD file buffer to the VCD file.

Related Verilog task: $dumpflush

Syntax

```
vcd flush
```

Arguments

None.

See also

See *Chapter 13 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.

vcd limit

The **vcd limit** command specifies the maximum size of 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: $dumplimit

Syntax

```
vcd limit
      <filesize>
```

Arguments

`<filesize>`
Specifies the maximum VCD file size in bytes. Required.

See also

See *Chapter 13 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.

vcd off

The **vcd off** command turns off VCD dumping and records all VCD variable values as x.

Related Verilog task: $dumpoff

Syntax

```
vcd off
```

Arguments

None.

See also

See *Chapter 13 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog tasks are documented in the IEEE 1364 standard.

vcd on

The **vcd on** command turns on VCD dumping and records the current values of all VCD variables. By default, **vcd on** is automatically performed at the end of the simulation time that the **vcd add** (CR-160) commands are performed.

Related Verilog task: $dumpon

Syntax

```
vcd on
```

Arguments

None.

See also

See *Chapter 13 - Value Change Dump (VCD) Files* for more information on VCD files. Verilog system tasks are documented in the IEEE 1364 standard.

vcom

The **vcom** command is used to invoke VCOM, the Model Technology VHDL compiler. Use VCOM to compile VHDL source code into a specified working library (or to the **work** library by default).

This command may be invoked from within ModelSim or from the operating system command prompt. This command may also be invoked during simulation.

Syntax

```
vcom
[-help] [-93] [-87] [-check_synthesiss] [-debugVA] [-explicit]
[-f <filename>] [-just eapbc] [-skip eapbc] [-line <number>]
[-no1164] [-noaccel <package_name>] [-nocheck]
[-nodebug[=ports]] [-nolock] [-nologo] [-novital] [-novitalcheck]
[-nowarn <number>] [-O0 | -O1 | -O4 | -O5] [-quiet] [-refresh] [-s]
[-source] [-version] [-work <library_name>] <filename>
```

Arguments

`-help`
Displays the command's options and arguments. Optional.

`-93`
Specifies that the simulator is to support VHDL 1076-1993. Optional. If used, must be the first argument. See additional discussion in the examples.

`-87`
Disables support for VHDL 1076-1993. This is the VCOM default. Optional. If used, must be the first argument. See additional discussion in the examples. Note that the default can be changed with the *modelsim.ini* file, see "Preference variables located in INI and MPF files" (B-394).

`-check_synthesis`
Turns on limited synthesis rule compliance checking. Optional. Checks to see that signals read by a process are in the sensitivity list.

`-debugVA`
Prints a confirmation if a VITAL cell was optimized, or an explanation of why it was not, during VITAL level-1 acceleration. Optional.

`-explicit`
Used to ignore an error in packages supplied by some other EDA vendors; directs the compiler to resolve ambiguous function overloading in favor of the explicit function definition. See additional discussion in the examples.

`-f <filename>`
Specifies a file with more command line arguments. Allows complex arguments to be reused without retyping. Optional.

`-just eapbc`
Directs the compiler to "just" include:

```
e - entities
a - architectures
p - packages
b - bodies
c - configurations
```

Any combination in any order can be used, but one choice is required if you use this optional switch.

`-skip eapbc`
Directs the compiler to skip all:

```
e - entities
a - architectures
p - packages
b - bodies
c - configurations
```

Any combination in any order can be used, but one choice is required if you use this optional switch.

`-line <number>`
Starts the compiler on the specified line in the VHDL source file. Optional; by default, the compiler starts at the beginning of the file.

`-no1164`
Causes the source files to be compiled without taking advantage of the built-in version of the IEEE **std_logic_1164** package. Optional. This will typically result in longer simulation times for VHDL programs that use variables and signals of type **std_logic**.

`-noaccel <package_name>`
Turns off acceleration of the specified package in the source code using that package.

`-nocheck`
Disables run time range checking. In some designs, this could result in a 2X speed increase. Optional.

`-nodebug[=ports]`
Hides the internal data of the compiled design unit. Optional. The design unit's source code, internal structure, signals, processes, and variables will not display in ModelSim's windows. In addition, none of the hidden objects may be accessed through the Dataflow window or with VSIM commands. This also means that you cannot set breakpoints or single step within this code. Don't compile with this switch until you're done debugging.

Note that this is not a speed switch like the "nodebug" option on many other products.

The optional **=ports** switch hides the ports for the lower levels of your design; it should only be used to compile the lower levels of the design. If you hide the ports of the top level you will not be able to simulate the design.

Note: **-nodebug** provides protection for proprietary model information. The Verilog 'protect compiler directive provides similar protection, but uses a Cadence encryption algorithm that is unavailable to Model Technology.

Design units or modules compiled with -nodebug can only instantiate design units or modules that are also compiled -nodebug.

See additional discussion in "Source code security and -nodebug" (E-443).

`-nolock`
Overrides the library lock file. The lock file prevents mulitple users from concurrently accessing the same library. If you are a single user, disabling the lock file should not present a problem. Optional. Default is locked.

`-nologo`
Disables startup banner. Optional.

`-novital`
Causes VCOM to use VHDL code for VITAL procedures rather than the accelerated and optimized timing and primitive packages built into the simulator kernel. Optional.

`-novitalcheck`
Disables VITAL95 compliance checking if you are using VITAL 2.2b. Optional.

`-nowarn <number>`
Selectively disables an individual warning message. Optional. Multiple **-nowarn** switches are allowed. Warnings may be disabled for all compiles via the *modelsim.ini* file, see the "[vcom] VHDL compiler control variables" (B-395).

The warning message numbers are:

```
1 = unbound component
2 = process without a wait statement
3 = null range
4 = no space in time literal
5 = multiple drivers on unresolved signal
6 = compliance checks
7 = optimization messages
```

`-O0 | -O1 | -O4 | -O5`

Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

Enable PE-level optimization with **-O1**. Optional.

Enable standard EE optimizations with **-O4**. Default.

Enable maximum optimization with **-O5**. Optional. Use caution with this switch. We recommend use of this switch with large sequential blocks only, other uses may signifcantly increase compile times. Optional.

`-quiet`

Disable 'loading' messages. Optional.

`-refresh`

Regenerates a library image. Optional. By default, the work library is updated; use **-work <library>** to update a different library. See **vcom** "Examples" (CR-173) for more information.

`-s`

Instructs the compiler not to load the **standard** package. Optional. This argument should only be used if you are compiling the **standard** package itself.

`-source`

Displays the associated line of source code before each error message that is generated during compilation. Optional; by default, only the error message is displayed.

`-version`

Returns the version of the compiler as used by the licensing tools, such as "Model Technology ModelSim EE vcom 5.3 DEV Compiler 1999.06 Jun 29 1999".

`-work <library_name>`

Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional; by default, the compiled design units are added to

the **work** library. The specified pathname overrides the pathname specified for work in the project file.

`<filename>`
Specifies the name of a file containing the VHDL source to be compiled. One filename is required; multiple filenames can be entered separated by spaces or wildcards may be used, i.e., "*.vhd". No switches can appear after filename(s) on the command line.

Examples

```
vcom example.vhd
```
The example compiles the VHDL source code contained in the file *example.vhd*.

```
vcom -87 o_units1 o_units2
```

```
vcom -93 n_unit91 n_unit92
```

Model*Sim* supports designs that use elements conforming to both the 1993 and the 1987 standards. Compile the design units separately using the appropriate switches.

Note that in the example above, the **-87** switch on the first line is redundant since the VCOM default is to compile to the 1987 standard.

```
vcom -nodebug example.vhd
```
Hides the internal data of *example.vhd*. Models compiled with **-nodebug** cannot use any of the Model*Sim* debugging features; any subsequent user will not be able to see into the model.

```
vcom -nodebug=ports level3.vhd levle2.vhd
vcom -nodebug top.vhd
```
The first line compiles and hides the internal data, plus the ports, of the lower-level design units, *level3.vhd* and *level2.vhd*. The second line compiles the top-level unit, *top.vhd*, without hiding the ports. It is important to compile the top level without **=ports** because top-level ports must be visible for simulation.

See "Source code security and -nodebug" (E-443) for more details.

```
vcom -noaccel numeric_std example.vhd
```
When compiling source that uses the **numeric_std** package, this command turns off acceleration of the **numeric_std** package, located in the **ieee** library.

```
vcom -explicit example.vhd
```
Although it is not intuitively 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, i.e.,

```
ARITHMETIC.”=”(left, right)
```

To eliminate that inconvenience, the VCOM command has the **-explicit** option that allows the explicit = operator to hide the implicit one. Allowing the explicit declaration to hide the implicit declaration is what most VHDL users expect.

```
vcom -work mylib -refresh
```

The **-work** option specifies **mylib** as the library to regenerate. **-refresh** rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of Model*Sim* (4.6 and later only).

If your library contains Verilog design units be sure to regenerate the library with **vlog** (CR-199) and **-refresh** as well.

See "Regenerating your design libraries" (2-41) for more information.

vdel

The **vdel** command deletes a design unit from a specified library.

Syntax

```
vdel
      [-help] [-verbose] [-lib <library_name>] [-all | <design_unit>
      [<arch_name>]
```

Arguments

`-help`
Displays the command's options and arguments. Optional.

`-verbose`
Displays progress message. Optional.

`-lib <library_name>`
Specifies the logical name or pathname of the library that holds the design unit to be deleted. Optional; by default, the design unit is deleted from the **work** library.

`-all`
Deletes an entire library. Optional. BE CAREFUL! Libraries cannot be recovered once deleted, and you are not prompted for confirmation.

`<design_unit>`
Specifies the entity, a package, configuration, or module to be deleted. Required unless **-all** is used.

`<arch_name>`
Specifies the name of an architecture to be deleted. Optional; if omitted, all of the architectures for the specified entity are deleted. Invalid for a configuration or a package.

Examples

`vdel -all`
Deletes the **work** library.

`vdel -lib synopsys -all`
Deletes the **synopsys** library.

`vdel xor`
Deletes the entity named **xor** and all its architectures from the **work** library.

`vdel xor behavior`
Deletes the architecture named **behavior** of the entity **xor** from the **work** library.

`vdel base`
Deletes the package named **base** from the **work** library.

vdir

The **vdir** command selectively lists the contents of a design library.

Syntax

```
vdir
      [-help] [-l] [-lib <library_name>] [<design_unit>]
```

Arguments

`-help`
Displays the command's options and arguments. Optional.

`-l`
Prints the version of **vcom** or **vlog** that each design unit was compiled under. Also prints the object-code version number that indicates which versions of **vcom/vlog** and Model*Sim* are compatible. This example was printed by **vdir -l** for the counter entity in the **work** library:

```
# ENTITY counter
# Source modified time: 930595550
# Source file: counter.vhd
# Version number: `0aC^@H>0f:X]@NeVdEN13
# Opcode format: 5.3 DEV; VCOM EE Object version 14
```

`-lib <library_name>`
Specifies the logical name or the pathname of the library to be listed. Optional; by default, the contents of the **work** library are listed.

`<design_unit>`
Indicates the design unit to search for within the specified library. If the design unit is a VHDL entity, its architectures are listed. Optional; by default, all entities, configurations, modules, and packages in the specified library are listed.

Example

```
vdir -lib design my_asic
```
Lists the architectures associated with the entity named **my_asic** that resides in the HDL design library called **design**.

vgencomp

Once a Verilog module is compiled into a library, you can use the **vgencomp** command to write its equivalent VHDL component declaration to standard output. Optional switches allow you to generate bit or vl_logic port types; std_logic port types are generated by default.

Syntax

```
vgencomp
      [-help] [-<library_name>] [-b] [-s]
      [-v] <module_name>
```

Arguments

`-help`
Displays the command's options and arguments. Optional.

`-<library_name>`
Specifies the pathname of the working library. If not specified, the default library **work** is used. Optional.

`-b`
Causes **vgencomp** to generate bit port types. Optional.

`-s`
Used for the explicit declaration of default std_logic port types. Optional.

`-v`
Causes **vgencomp** to generate vl_logic port types. Optional.

`<module_name>`
Specifies the name of Verilog module to be accessed. Required.

Examples

This example uses a Verilog module that is compiled into the **work** library. The module begins as Verilog source code:

```
module top(i1, o1, o2, io1);
    parameter width = 8;
    parameter delay = 4.5;
    parameter filename = "file.in";

    input i1;
    output [7:0] o1;
    output [4:7] o2;
    inout [width-1:0] io1;
endmodule
```

After compiling, **vgencomp** is invoked on the compiled module:

```
vgencomp top
```

and writes the following to stdout:

```
component top

    generic(
        width           : integer := 8;
        delay           : real    := 4.500000;
        filename        : string  := "file.in"
            );
    port(
        i1              : in    std_logic;
        o1              : out   std_logic_vector(7 downto 0);
        o2              : out   std_logic_vector(4 to 7);
        io1             : inout std_logic_vector
        );
end component;
```

view

The **view** command will open a Model*Sim* window and bring that window to the front of the display. If multiple instances of a window exist, **view** will change the default window of that type to the specified window. Using the **-new** option, **view** will create an additional instance of the specified window type and set it to be the default window for that type.

Names for windows are generated as follows:

- The first window name (automatically created without using **-new**) has the same name as the window type.
- Additional window names created by **-new** append an integer to the window type, starting with 1.

Syntax

```
view
      [*] [-x <n>] [-y <n>] [-height <n>] [-width <n>] [-icon]
      [-new] <window_name> ...
```

Arguments

*****
Wildcards can be used, for example: l* (List window), s* (Signal, Source, and Structure windows), even * alone (all windows). Optional.

-x <n>
Specify the window upper-left-hand x-coordinate in pixels. Optional

-y <n>
Specify the window upper-left-hand y-coordinate in pixels. Optional

-height <n>
Specify the window height in pixels. Optional

-width <n>
Specify the window width in pixels. Optional

-icon
Toggles the view between window and icon. Optional

`-new`

Creates a new instance of the window type specified with the **<window_name>** option. New window names are created by appending an integer to the window type, starting with 1, then incrementing the integer.

`<window_name> ...`

Specifies the Model*Sim* window type to view. Multiple window types may be used; at least one type (or wildcard) is required. Available window types are:

```
dataflow, list, process, signals, source, structure, variables, and
wave
```

Also creates a new instance of the specified window type when used with the **-new** option. You may also specify the window(s) to view when multiple instances of that window type exist, i.e., wave2 structure1.

Examples

`view wave`

Creates a window named ‘wave’. Its full Tk path is ‘.wave’.

`view -new wave`

Creates a window named ‘wave1’. Its full Tk path is ‘.wave1’. Wave1 is now the default Wave window. Any **add wave** command (CR-33) would add items to wave1.

`view wave`

Changes the default window back to ‘wave’.

`add wave -win .wave1 mysig`

Will override the default window and add *mysig* to wave1.

See also

add wave command (CR-33)

virtual delete|describe|define

The **virtual delete** command removes the matching virtuals.

The **virtual describe** command prints to the transcript window a complete description of the data type of one or more virtual signals. Similar to the existing **describe** command.

The **virtual define** command prints to the transcript window the definition of the virtual signal or function in the form of a command that can be used to re-create the object.

Syntax

```
virtual delete|describe|define
      [-kind <kind>] <path>|<wildcard>
```

Arguments

`-kind<kind>`
A subset of virtuals to look at when wildcards are used. Optional. **<kind>** can be any of the following: signals, functions, implicits, explicits (unique abbreviations are accepted).

Examples

```
virtual define -kind explicits *
```

Shows the definitions of all the virtuals you have explicitly created.

See also

virtual signal command (CR-193), **virtual function** command (CR-184), **virtual region** command (CR-190), **virtual expand** command (CR-183), **vmap** command (CR-207), **virtual log|nolog** command (CR-189), **virtual save** command (CR-191), **virtual show|count** command (CR-192), **virtual type** command (CR-196), **searchLog** command (CR-144)

virtual expand

The **virtual expand** command produces a list of all the non-virtual objects contained in the virtual signal(s). This can be used to create a list of arguments for a command that does not accept or understand virtual signals.

Syntax

```
virtual expand
```

[-base] [<path>|<wildcard>]+

Arguments

`-base`

Causes the root signal parent to be output in place of a subelement. Optional. So for:

```
power add [virtual expand -base myVirtualSignal]
```

the resulting command after substitution would be:

```
power add signala signalb signalc
```

Examples

```
power add [virtual expand myVirtualSignal]
```

Adds the elements of a virtual signal to the power reporting tool.

In the Tcl language, the square brackets specify that the enclosed command should be executed first ("virtual expand ..."), then the result substituted into the surrounding command. So if myVirtualSignal is a contatenation of signala, signalb.rec1 and signalc(5 downto 3), the resulting command after substitution would be:

```
power add signala signalb.rec1 {signalc(5 downto 3)}
```

The slice of signalc is quoted in curly braces automatically because it contains spaces.

See also

virtual signal command (CR-193), **virtual function** command (CR-184), **virtual region** command (CR-190), **virtual delete|describe|define** command (CR-182), **vmap** command (CR-207), **virtual log|nolog** command (CR-189), **virtual save** command (CR-191), **virtual show|count** command (CR-192), **virtual type** command (CR-196), **searchLog** command (CR-144)

virtual function

The **virtual function** command creates a new signal, known only by the GUI (not the kernel), that consists of logical operations on existing signals and simulation time, as described in **<expressionString>**. It can handle bit selects and slices of Verilog registers.

The virtual function will show up in the wave and signals window as an expandible object if it references more than a single scalar signal. The children correspond to the inputs of the virtual function. This allows the virtual 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 be used to gate the list window display.

Syntax

```
virtual function
      [-env <path>] [-install <path>] [-implicit] {<expressionString>}
      <name>
```

Arguments

(Arguments for **virtual function** are the same as those for **virtual signal** except for the contents of the expression string.)

`-env <path>`
Specifies a hierarchical context for the signal names in **<expressionString>** so they don't all have to be full paths. Optional.

`-install <path>`
Causes the newly-created signal to become a child of the specified region. If **-install** is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in **<expressionString>**. If the expression references more than one logfile (dataset), the virtual signal will automatically be placed in region virtuals:/Signals. Optional.

`-implicit`
Used internally to create virtuals that are automatically saved with the List or Wave format. Optional.

`<expressionString>`
A text string expression in the MTI GUI expression format. Required. See (TBD) for a full and updated description.

`<name>`
The name you define for the virtual signal. Required. Case is ignored unless installed in a Verilog region. It is recommended to use alpha, numeric and underscore characters only. Or, use VHDL extended identifier notation, in which case **<name>** needs to be quoted with double quotes or with curly braces.

Examples

```
virtual function { not /chip/section1/clk } clk_n
```

Creates a signal /chip/section1/clk_n which is the inverse of /chip/section1/clk.

```
virtual function -install /chip { (std_logic_vector) chip.vlog.rega } rega_slv
```

Creates a std_logic_vector equivalent of a verilog register "rega" and installs it as /chip/rega_slv.

```
virtual function { /chip/addr[11:0] == 0xfab } addr_eq_fab
```

Creates a boolean signal /chip/addr_eq_fab that is true when /chip/addr[11:0] is equal to hex "fab", and false otherwise. It is ok to mix VHDL signal path notation with Verilog part-select notation.

```
virtual function { gate:/chip/siga XOR rtl:/chip/siga) } siga_diff
```

Creates a signal that is non-zero only high during times at which a signal /chip/siga of the gate-level version of a design does not match /chip/siga of the rtl version of a 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/outbus XOR rtl:/chip/outbus) } outbus_diff
```

This example creates a one-bit signal "outbus_diff" which is non-zero during times at which any bit of a vector signal /chip/outbus of the gate-level version of a design does not match the corresponding bit of the signal in the rtl version of a design.

This expression uses the "OR-reduction" operator, which takes the logical OR of all the bits of the vector argument.

The following is an example of a series of commands that reconstruct a bus in a gate level design, and creates a comparison between a gate and RTL design:

```
virtual signal -env gate:chip { (range 7:0)&{a_07, a_06, a_05, a_04, a_03, a_02,
```

```
a_01, a_00} } a

virtual function { | (gate:/chip/a XOR rtl:/chip/a) } a_err

virtual function { | (gate:/chip/b XOR rtl:/chip/b) } b_err

...

virtual function { a_err OR b_err OR c_err .... } any_error
```

To find the differences, you can do any of the following:

a) Add "any_error" to the wave window and tab to the transitions

b) Use the "searchLog" command to print out the time/delta of the first error:

```
searchLog {0 ns} any_error
```

(See the "searchLog" command description below.)

c) Trigger the list window from the "any_error" signal. This will enable you to see in the list window ONLY the times at which errors occur. For example:

```
configure list -usegating 1
configure list -gateexpr { any_error == '1' }
```

(A future release will make signal comparison more automatic to set up and will allow comparisons at specified sampling points in time.)

Commands fully compatible with virtual functions

add list (CR-22)	add log /log (CR-93)	add wave (CR-33)
checkpoint (CR-53)	delete (CR-62)	describe (CR-63) ("virtual describe" is a little faster)
down | up (CR-70)	examine (CR-81)	find (CR-85)
restart (CR-133)	restore (CR-134)	right | left (CR-136)
search and next (CR-141)	searchLog (CR-144)	show (CR-148)

Commands not currently compatible with virtual functions

force (CR-87)	check contention add (CR-45)	check contention config (CR-46)
check contention off (CR-47)	check float add (CR-48)	check float config (CR-49)
check float off (CR-50)	check stable on (CR-51)	check stable off (CR-52)
drivers (CR-73)	noforce (CR-101)	power add (CR-111)
power report (CR-112)	power reset (CR-113)	toggle add (CR-154)
toggle reset (CR-155)	toggle report (CR-156)	vcd add (CR-160)
when (CR-226)		

See also

virtual signal command (CR-193), **virtual region** command (CR-190), **virtual delete|describe|define** command (CR-182), **virtual expand** command (CR-183), **vmap** command (CR-207), **virtual log|nolog** command (CR-189), **virtual save** command (CR-191), **virtual show|count** command (CR-192), **virtual type** command (CR-196), **searchLog** command (CR-144)

virtual hide|nohide

The **virtual hide** command sets a flag in the specified real or virtual signals so that the signals do not appear in the Signals window. This is used when you want to replace a bit-blasted buss with a user-defined bus.

The **virtual nohide** command resets the flag in the specified signal objects.

Syntax

```
virtual hide|nohide
      [-kind <virtualKind>]|[-region <path>] <signalName>|<pattern>
```

Arguments

`-kind<virtualKind>`
Used to limit the scope to a particular kind of virtuals. Optional. For instance, you can use it to hide all the explicit virtuals, no matter where they are installed in the design hierarchy.

`-region<path>`
Used in place of **-kind** to specify a region of design space to look for the signal names. Optional

`<signalName>|<pattern>`
Indicates which signal names or wildcard pattern should be used in finding the signals to hide or expose. Required. Any number of names or wildcard patterns may be used.

See also

virtual signal command (CR-193), **virtual function** command (CR-184), **virtual region** command (CR-190), **virtual delete|describe|define** command (CR-182), **virtual expand** command (CR-183), **virtual log|nolog** command (CR-189), **virtual save** command (CR-191), **virtual show|count** command (CR-192), **virtual type** command (CR-196), **searchLog** command (CR-144)

virtual log|nolog

The **virtual log** command causes the sim-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.

The **virtual nolog** command causes the sim-mode dependent signals of the specified virtual signals to be un-logged by the kernel. If wildcard patterns are used, it will also log any normal signals found, unless the **-only** option is used.

The action of the **virtual log|nolog** command can also be obtained by using the **log -virtual** command or the **nolog -virtual** command. Also, the log and nolog commands have a new **-virtualOnly** option that does the same as the **-only** option below.

Syntax

```
virtual log|nolog
      [-kind <virtualKind>]|[-region <path>] [-recursive] [-only] [-in]
      [-out] [-inout] [-internal] [-ports] [<signalName>|<pattern>]+
```

Arguments

`-only`
Can be used with a wildcard to specify that only the virtual signals found by the wildcard should be logged or unlogged. Optional.

(Other options are the same as in other virtual or log/nolog commands)

See also

virtual signal command (CR-193), **virtual function** command (CR-184), **virtual region** command (CR-190), **virtual delete|describe|define** command (CR-182), **virtual expand** command (CR-183), **vmap** command (CR-207), **virtual save** command (CR-191), **virtual show|count** command (CR-192), **virtual type** command (CR-196), **searchLog** command (CR-144)

virtual region

The **virtual region** command creates a new user-defined design hierarchy region.

Syntax

```
virtual region
      <parentPath> <regionName>
```

Arguments

`<parentPath>`
The full path to the region that will become the parent of the new region. Required.

`<regionName>`
The name you give to the new region. Required

See also

virtual signal command (CR-193), **virtual function** command (CR-184), **virtual delete|describe|define** command (CR-182), **virtual expand** command (CR-183), **vmap** command (CR-207), **virtual log|nolog** command (CR-189), **virtual save** command (CR-191), **virtual show|count** command (CR-192), **virtual type** command (CR-196), **searchLog** command (CR-144)

virtual save

The **virtual save** command saves the definitions of virtuals to a file.

Syntax

```
virtual save
      [-kind <kind>] [-append] [<filename>]
```

Arguments

`-kind<kind>`
Specifies a subset of virtuals to save. Optional.

`-append`
Specifies to save ONLY THOSE VIRTUALS that are neither already saved nor those that were read in from a macro file. These unsaved virtuals are then appended to the specified or default file. Optional.

`<filename>`
Used for writing the virtual definitions. Optional. If **<filename>** is not specified, the default virtual filename will be used, which can be specified in the pref.tcl file, and defaults to *virtuals.do*.

See also

virtual signal command (CR-193), **virtual function** command (CR-184), **virtual region** command (CR-190), **virtual delete|describe|define** command (CR-182), **virtual expand** command (CR-183), **vmap** command (CR-207), **virtual log|nolog** command (CR-189), **virtual show|count** command (CR-192), **virtual type** command (CR-196), **searchLog** command (CR-144)

virtual show|count

The **virtual show** command lists the full path names of all the virtuals explicitly defined.

The **virtual count** command counts the number of explicitly declared virtuals that have not been saved and that were not read in using a macro file. These are virtuals that probably need to be saved before Model*Sim* quits.

Syntax

```
virtual show|count
      [-kind <kind>]
```

Arguments

`-kind<kind>`
Specifies a subset of virtuals to look at. Optional.

See also

virtual signal command (CR-193), **virtual function** command (CR-184), **virtual region** command (CR-190), **virtual delete|describe|define** command (CR-182), **virtual expand** command (CR-183), **vmap** command (CR-207), **virtual log|nolog** command (CR-189), **virtual save** command (CR-191), **virtual type** command (CR-196), **searchLog** command (CR-144)

virtual signal

The **virtual signal** command creates a new signal, known only by the GUI (not the kernel), that consists of concatenations of signals and subelements as specified in **<expressionString>**. It cannot handle bit selects and slices of Verilog registers.

Syntax

```
virtual signal
      [-env <path>] [-install <path>] [-implicit] {<expressionString>}
      <name>
```

Arguments

`-env <path>`
Specifies a hierarchical context for the signal names in **<expressionString>** so they don't all have to be full paths. Optional.

`-install <path>`
Causes the newly-created signal to become a child of the specified region. If **-install** is not specified, the newly-created signal becomes a child of the nearest common ancestor of all objects appearing in **<expressionString>**. If the expression references more than one logfile (dataset), the virtual signal will automatically be placed in region virtuals:/Signals. Optional.

`-implicit`
Used internally to create virtuals that are automatically saved with the List or Wave format. Optional.

`<expressionString>`
A text string expression in the MTI GUI expression format. Required. See (TBD) for a full and updated description.

`<name>`
The name you define for the virtual signal. Required. Case is ignored unless installed in a Verilog region. It is recommended to use alpha, numeric and underscore characters only. Or, use VHDL extended identifier notation, in which case **<name>** needs to be quoted with double quotes or with curly braces.

Examples

```
virtual signal -env sim:/chip/alu { (range 4 downto 0)(a_04 & a_03 & a_02 & a_01
```

```
& a_00) } a
```

This command reconstructs a bus "sim:/chip/alu/a(4 downto 0)", using VHDL notation, assuming that a_ii are scalars all of the same type.

```
virtual signal -env sim:chip.alu { (range 4:0)&{a_04, a_03, a_02, a_01, a_00} } a
```

This command reconstructs a bus "sim:chip.alu.a[4:0]", using Verilog notation. Note that the concatenation notation starts with "&{" rather than "{".

```
virtual signal -install sim:/testbench { /chipa/alu/a(19 downto 13) & \ /chipa/
decode/inst & /chipa/mode } stuff
```

Assuming /chipa/mode is of type integer and /chipa/alu/a is of type std_logic_vector, and /chipa/decode/inst is a user-defined enumeration, this example creates a signal sim:/testbench/stuff which is a record type with three fields corresponding to the three specified signals.

```
virtual signal { chip.instruction[23:21] } address_mode
```

This creates a three-bit signal, chip.address_mode, as an alias to the specified bits.

Commands fully compatible with virtual signals

add list (CR-22)	add log / log (CR-93)	add wave (CR-33)
checkpoint (CR-53)	delete (CR-62)	describe (CR-63) ("virtual describe" is a little faster)
down | up (CR-70)	examine (CR-81)	find (CR-85)
force (CR-87)/noforce (CR-101)	restart (CR-133)	restore (CR-134)
right | left (CR-136)	search and next (CR-141)	searchLog (CR-144)
show (CR-148)		

Commands compatible with virtual signals using [virtual expand <signal>]

check contention add (CR-45)	check contention config (CR-46)	check contention off (CR-47)
check float add (CR-48)	check float config (CR-49)	check float off (CR-50)
check stable on (CR-51)	check stable off (CR-52)	drivers (CR-73)
power add (CR-111)	power report (CR-112)	power reset (CR-113)
toggle add (CR-154)	toggle reset (CR-155)	toggle report (CR-156)
vcd add (CR-160)		

Commands not currently compatible with virtual signals

when (CR-226)

See also

virtual function command (CR-184), **virtual region** command (CR-190), **virtual delete|describe|define** command (CR-182), **virtual expand** command (CR-183), **vmap** command (CR-207), **virtual log|nolog** command (CR-189), **virtual save** command (CR-191), **virtual show|count** command (CR-192), **virtual type** command (CR-196), **searchLog** command (CR-144)

virtual type

The **virtual type** command creates a new enumerated type, known only by the GUI, not the kernel. Virtual types are used to convert signal values to character strings.

Syntax

```
virtual type
      {<list_of_strings>} <name>
```

Arguments

`{<list_of_strings>}`
A space-separated list of character strings. Required. Each string is associated with an enumeration index, starting at zero and increasing by one in the positive direction. There is currently no restriction on the contents of each string, but if strings contain spaces they would need to be quoted, and if they contain characters treated specially by Tcl (square brackets, curly braces, backslashes...), they would need to be quoted with curly braces.

`<name>`
The user-defined name of the virtual type. Required. Case is not ignored. It is recommended to use alpha, numeric and underscore characters only, or use VHDL extended identifier notation, in which case **<name>** needs to be quoted with double quotes or with curly braces.

Examples

Examples of virtual types and their use in displaying signals:

```
virtual type {state0 state1 state2 state3} mystateType
virtual function {(mystateType)mysignal} myConvertedSignal
```

When **myConvertedSignal** is displayed in the Wave, List or Signals window, the string "state0" will appear when **mysignal** == 0, and "state1" when **mysignal** == 1, and "state2" when **mysignal** == 2, etc.

Signal **mysignal** may be a VHDL array of std_logic/std_ulogic/bit, or a Verilog register or vector net, or a VHDL integer.

```
virtual type {idle reading writing waiting} mystateType
virtual signal {instruction[19:18]} stateDecode
virtual function {(mystateType)stateDecode} myConvertedState
add wave myConvertedState
```

When picking out a slice and converting the slice, it works best to do it in two steps -- first define the slice, then convert the slice signal to the enumerated type.

See also

virtual function command (CR-184), **virtual region** command (CR-190), **virtual delete|describe|define** command (CR-182), **virtual expand** command (CR-183), **vmap** command (CR-207), **virtual log|nolog** command (CR-189), **virtual save** command (CR-191), **virtual show|count** command (CR-192), **virtual signal** command (CR-193), **searchLog** command (CR-144)

vlib

The **vlib** command creates a design library.

Syntax

```
vlib
      [-help] [-dos| -short] [-unix | -long] <directory_name>
```

Arguments

`-help`
Displays the command's options and arguments. Optional.

`-dos`
Specifies that subdirectories in a library have names that are compatible with DOS. Not recommended if you use the **vmake** (CR-205) utility. Optional. Default for Model*Sim* PE.

`-short`
Interchangeable with the **-dos** argument. Optional.

`-unix`
Specifies that subdirectories in a library may have long file names that are NOT compatible with DOS. Optional. Default for Model*Sim* EE.

`-long`
Interchangeable with the **-unix** argument. Optional.

`<directory_name>`
Specifies the pathname of the library to be created. Required.

Examples

`vlib design`
Creates the design library called **design**. You can define a logical name for the library using the **vmap** command (CR-207) or by adding a line to the library section of the *modelsim.ini* file that is located in the same directory.

The **vlib** command must be used to create a library directory. Operating system commands cannot be used to create a library directory or index file.

If the specified library already exists as a valid Model*Sim* library, the **vlib** command will exit with an error message without touching the library.

vlog

The **vlog** command is used to invoke VLOG, the Model Technology Verilog compiler. Use **vdel** (CR-175) to compile Verilog source code into a specified working library (or to the **work** library by default). The **vlog** command is used just like **vdel** (CR-175), except that you do not need to compile a module before it is referenced (unless the module is referenced from VHDL).

This command may be invoked from within ModelSim or from the operating system command prompt. This command may also be invoked during simulation.

Syntax

```
vlog
      [-help] [-compat] [+define+<macro_name>[=<macro_text>]]
      [+delay_mode_distributed] [+delay_mode_path] [+delay_mode_unit]
      [+delay_mode_zero] [-f <filename>][-hazards] [+incdir+<directory>]
      [-incr] [+libext+<suffix>] [+librescan] [-line <number>]
      [+mindelays] [+maxdelays] [-nodebug[=ports | =pli]] [+nolibcell]
      [-noincr] [-nolock] [-nologo] [-O0 | -O1 | -O4 | -O5] [-quiet] [-R
      <simargs>] [-refresh] [-source] [+typdelays] [-u] [-v
      <library_file>] [-version] [-work <library_name>] [-y
      <library_directory>] [-93] <filename>
```

Arguments

`-help`
Displays the command's options and arguments. Optional.

`-compat`
The Verilog language does not specify the order that a simulator must execute simultaneous events; however, some models depend on the event ordering of the simulator that the model was developed on. The **-compat** switch disables optimizations that result in an event order that is different from some other widely used Verilog simulators. You can also use the **-hazards** switch to help find code that depends on a specific event ordering.

`+define+<macro_name>[=<macro_text>]`
Same as compiler directive: **'define macro_name macro_text**. Optional.

`+delay_mode_distributed`
Use structural delays and ignore path delays. Optional.

`+delay_mode_path`
Set structural delays to zero and use path delays. Optional.

`+delay_mode_unit`
Set non-zero structural delays to one. Optional.

`+delay_mode_zero`
Set structural delays to zero. Optional.

`-f <filename>`
Specifies a file with more command line arguments. Allows complex arguments to be reused without retyping. Optional.

`-hazards`
Enables the run-time hazard checking code. Optional.

`+incdir+<directory>`
Search directory for files included with the **'include filename** compiler directive. Optional.

`-incr`
Performs incremental compile. Optional. Compiles only code that has changed, or if compile options change.

`+libext+<suffix>`
Specifies the suffix of files in library directory. Multiple suffixes may be used, for example: **+libext+.v+.u**. Optional.

`+librescan`
Scan libraries in command-line order for all unresolved modules. Optional.

`-line <number>`
Specify the starting line number. Optional.

`+mindelays`
Selects minimum timing from Verilog **min:typ:max** expressions. Optional.

`+maxdelays`
Selects maximum timing from Verilog **min:typ:max** expressions. Optional.

`-noincr`
Disables incremental compile previously turned on with **-incr**. Optional.

`-nodebug[=ports | =pli]`
Hides the internal data of the compiled design unit. Optional. The design unit's source code, internal structure, signals, processes, and variables will not display in Model*Sim*'s windows. In addition, none of the hidden objects may be accessed through the Dataflow window or with VSIM commands. This also means that you

cannot set breakpoints or single step within this code. Don't compile with this switch until you're done debugging.

Note that this is not a speed switch like the "nodebug" option on many other products.

The optional **=ports** switch hides the ports for the lower levels of your design; it should only be used to compile the lower levels of the design. If you hide the ports of the top level you will not be able to simulate the design.

The optional **=pli** switch prevents the use of pli functions to interrogate individual modules for information; this switch may be used at any level of the design. Combine both switches with **=ports+pli** or **=pli+ports**.

Note: **-nodebug** provides protection for proprietary model information. The Verilog 'protect compiler directive provides similar protection, but uses a Cadence encryption algorithm that is unavailable to Model Technology.

See additional discussion in "Source code security and -nodebug" (E-443).

`+nolibcell`
Do not automatically define library modules as cells. Optional.

`-nolock`
Overrides the library lock file. The lock file prevents mulitple users from concurrently accessing the same library. If you are a single user, disabling the lock file should not present a problem. Optional. Default is locked.

`-O0 | -O1 | -O4 | -O5`
Lower the optimization to a minimum with **-O0** (capital oh zero). Optional. Use this to work around bugs, increase your debugging visibility on a specific cell, or when you want to place breakpoints on source lines that have been optimized out.

Enable PE-level optimization with **-O1**. Optional.

Enable standard EE optimizations with **-O4**. Default.

Enable maximum optimization with **-O5**. Optional. Use caution with this switch. We recommend use of this switch with large sequential blocks only, other uses may signifcantly increase compile times. Optional.

`-quiet`
Disables 'loading' messages. Optional.

`-R <simargs>`
Causes VSIM to be invoked on the top-level Verilog modules immediately following compilation. VSIM is invoked with the arguments specified by **<simargs>** (any arguments available for **vsim** (CR-208)).

`-refresh`
Regenerates a library image. Optional. By default, the work library is updated; use **-work <library>** to update a different library. See **vlog** examples for more information.

`-source`
Displays the associated line of source code before each error message that is generated during compilation. Optional; by default, only the error message is displayed.

`+typdelays`
Selects typical timing from Verilog **min:typ:max** expressions. Optional. Default.

`-u`
Converts regular Verilog identifiers to uppercase. Allows case insensitivity for module names. Optional.

`-v <library_file>`
Specifies the Verilog source library file to search for undefined modules. Optional. 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. See additional discussion in the examples.

`-version`
Returns the version of the compiler as used by the licensing tools, such as "Model Technology ModelSim EE vlog 5.3 Compiler 1999.06 Jul 2 1999".

`-work <library_name>`
Specifies a logical name or pathname of a library that is to be mapped to the logical library **work**. Optional; by default, the compiled design units are added to the **work** library. The specified pathname overrides the pathname specified for work in the project file.

`-y <library_directory>`
Specifies the Verilog source library directory to search for undefined modules. Optional. 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. You will need to specify a file suffix by using **-y** in conjunction with the **+libext+<suffix>** option if your filenames differ from your module names. See additional discussion in the examples.

`-93`
Specifies that the VHDL interface to Verilog modules shall use VHDL 1076-93 extended identifiers to preserve case in Verilog identifiers that contain uppercase letters.

`<filename>`
Specifies the name of the Verilog source code file to compile. One filename is required. Multiple filenames can be entered separated by spaces.

Examples

`vlog example.vlg`
The example compiles the Verilog source code contained in the file *example.vlg*.

`vlog -nodebug example.v`
Hides the internal data of *example.v*. Models compiled with **-nodebug** cannot use any of the Model*Sim* debugging features; any subsequent user will not be able to see into the model.

`vlog -nodebug=ports level3.v levle2.v`

`vcom -nodebug top.v`
The first line compiles and hides the internal data, plus the ports, of the lower-level design units, *level3.v* and *level2.v*. The second line compiles the top-level unit, *top.v*, without hiding the ports. It is important to compile the top level without **=ports** because top-level ports must be visible for simulation.

`vlog -nodebug=ports+pli level3.v level2.v`

`vlog -nodebug=pli top.v`
The first command hides the internal data, and ports of the design units, *level3.v* and *level2.v*. In addition it prevents the use of pli functions to interrogate the compiled modules for information (either **=ports+pli** or **=pli+ports** works fine for this command). The second line compiles the top-level unit without hiding the ports but restricts the use of pli functions as well.

Note that the **=pli** switch may be used at any level of the design but **=ports** should only be used on lower levels since you can't simulate without visible top-level ports.

See "Source code security and -nodebug" (E-443) for more details.

`vlog top.v -v und1`

After compiling *top.v*, **vlog** will scan the file *und1* for modules or primitives referenced but undefined in *top.v*. Only referenced definitions will be compiled.

`vlog top.v +libext+.v+.u -y vlog_lib`

After compiling *top.v*, **vlog** will scan the **vlog_lib** library for files with modules with the same name as primitives referenced, but undefined in *top.v*. The use of **+libext+.v+.u** implies filenames with a *.v* or *.u* suffix (any combination of suffixes may be used). Only referenced definitions will be compiled.

`vlog -work mylib -refresh`

The **-work** option specifies **mylib** as the library to regenerate. **-refresh** rebuilds the library image without using source code, allowing models delivered as compiled libraries without source code to be rebuilt for a specific release of Model*Sim* (4.6 and later only).

If your library contains VHDL design units be sure to regenerate the library with **vdel** (CR-175) using the **-refresh** option as well. See "Regenerating your design libraries" (2-41) for more information.

`vlog module1.v -u -O0 -incr`

The **-incr** option determines whether or not the module source or compile options have changed as module1 is parsed. If no change is found, the code generation phase is skipped. Differences in compile options are determined by comparing the compiler options stored in the _info file with the compiler options given. They must match exactly.

vmake

The **vmake** utility allows you to use a UNIX or Windows MAKE program to maintain libraries. The **vmake** utility is run on a compiled design library, and outputs a makefile that can be used to reconstruct the library. The resulting makefile can then be run with a version of MAKE (not supplied with Model*Sim*); a MAKE program is included with Microsoft's Visual C/C++, as well as many other program development environments.

After running the **vmake** utility, MAKE will recompile only the design units (and their dependencies) that have changed. **Vmake** only needs to be run 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.

This command must be invoked from either the UNIX or the Windows/DOS prompt.

Syntax

```
vmake
      [-help] [<library_name>] [><makefile>]
```

Arguments

`-help`
Displays the command's options and arguments. Optional.

`<library_name>`
Specifies the library name; if none is specified, then **work** is assumed. Optional.

`><makefile>`
Specifies the makefile name. Optional.

Examples

To produce a makefile for the work library:

```
vmake >makefile
```

You can also run **vmake** on libraries other than **work**:

```
vmake mylib >mylib.mak
```

To rebuild **mylib,** specify its makefile when you run MAKE:

```
make -f mylib.mak
```

vmap

The **vmap** command defines a mapping between a logical library name and a directory by modifying the *modelsim.ini* file. With no arguments, reads the appropriate *modelsim.ini* file(s) and prints the current VHDL logical library to physical directory mappings. Returns nothing.

Syntax

```
vmap
      [-help] [-c] [-del] [<logical_name>] [<path>]
```

Arguments

`-help`
Displays the command's options and arguments. Optional.

`-c`
Copies the default *modelsim.ini* file from the Model*Sim* installation directory to the current directory. Optional.

`-del`
Deletes the mapping specified by <logical_name> from the current project file. Optional.

`<logical_name>`
Specifies the logical name of the library to be mapped. Optional.

`<path>`
Specifies the pathname of the directory to which the library is to be mapped. Optional. If omitted, the command displays the mapping of the specified logical name.

vsim

The **vsim** command is used to invoke the VSIM simulator, or to view the results of a previous simulation run (when invoked with the **-view** switch). You can specify a configuration, an entity/architecture pair, or a module for simulation. If a configuration is specified, it is invalid to specify an architecture. With no options, **vsim** brings up the Load Design dialog box, allowing you to specify the design and options; the Load Design dialog box will not be presented if you specify any options. During elaboration **vsim** determines if the source has been modified since the last compile.

To **manually interrupt design elaboration** use the Break key or <control-c> (while the mouse cursor is located in the window that invoked **vsim** if you are running UNIX).

The **vsim** command may also be invoked from the command line within Model*Sim* with most of the options shown below (all except the **vsim -c** and **-restore** options).

Syntax

```
vsim
      [-c] [-coverage] [-do "<command_string>" | <macro_file_name>]
      [-f <filename>] [-gui] [-help] [-i] [-installcolormap] [-
      keepstdout]
      [-l <filename>]  [<license_option>]
      [-multisource_delay min | max | latest]
      [-nocompress] [+no_notifier] [+no_tchk_msg] [+notimingchecks]
      [-quiet] [-restore <filename>]
      [-sdfmin | -sdftyp | -sdfmax [<instance>=]<sdf_filename>]
      [-sdfnoerror] [-sdfnowarn] [-t [<multiplier>]<time_unit>]
      [-tag <string>] [-title <title>] [-trace_foreign <int>] [-version]
      [-view <filename>] [-wav <filename>] [-wavslim <size>]
      [-wavtlim <duration>]
      [-foreign <attribute>] [-g<Name=Value> ...] [-G<Name=Value> ...]
      [-nocollapse] [-noglitch] [+no_glitch_msg] [-std_input <filename>]
      [-std_output <filename>] [-strictvital] [-vcdread <filename>]
      [-vital2.2b]
      [+alt_path_delays] [-hazards] [-L <library_name> ...]
      [-Lf <library_name> ...] [+mindelays] [+maxdelays][+no_pulse_msg]
      [+nosdfwarn] [+nosdferror] [+nowarn<CODE>] [-pli "<object list>"]
      [+<plusarg>] [+pulse_e/<percent>] [+pulse_r/<percent>]
      [+typdelays]
      [<library_name>.<design_unit>]
```

VSIM arguments are grouped alphabetically by language:

- Arguments, VHDL and Verilog (CR-209)
- Arguments, VHDL (CR-213)
- Arguments, Verilog (CR-216)
- Arguments, design-unit (CR-217)

Arguments, VHDL and Verilog

`-c`
Specifies that the simulator is to be run in command line mode. Optional. If used, must be the first argument. Also see "Running command-line and batch-mode simulations" (E-440) for more information.

`-coverage`
Allows line number execution statistics to be kept by the simulator. By default no statistics are kept. This switch must be specified during command-line invocation of the simulator in order to use the various coverage commands: **coverage clear** (CR-59), **coverage reload** (CR-60), and **coverage report** (CR-61). Also see *Chapter 8 - ModelSim SE Code Coverage* for more information. Optional.

`-do "<command_string>" | <macro_file_name>`
Instructs VSIM to use the command(s) specified by <command_string> or the macro file named by <macro_file_name> rather than the startup file specified in the *.ini* file, if any. Optional.

`-f <filename>`
Specifies a file with more command line arguments. Allows complex arguments to be reused without retyping. Optional.

`-gui`
Starts the Model*Sim* GUI without loading a design. Optional.

`-help`
Displays the command's options and arguments. Optional.

`-i`
Specifies that the simulator is to be run in interactive mode. Optional. If used, must be the first argument.

`-installcolormap`
For UNIX only. Causes **vsim** to use its own colormap so as not to hog all the colors on the display. This is similar to the -install switch on Netscape. Optional.

`-keepstdout`
For use with foreign programs. Instructs the simulator to not redirect the stdout stream to the Main transcript window. Optional.

`-l <filename>`
Saves the contents of the "Main window" (10-158) to <filename>. Optional. Default is *transcript*. Can also be specified using the *.ini* (see "Creating a transcript file" (B-404)) file or the *.tcl* preference file.

`<license_option>`
Restricts the search of the license manager. Optional. Use one of the following options.

<license_option>	Description
-lic_nomgc	excludes any MGC licenses from the search
-lic_nomti	excludes any MTI licenses from the search
-lic_vlog	searches only for Model*Sim* EE/VLOG licenses
-lic_vhdl	searches only for Model*Sim* EE/VHDL licenses
-lic_plus	searches only for Model*Sim* EE/PLUS licenses
-lic_noqueue	do not wait in queue when license is unavailable

The options may also be specified with the License (B-399) in the *modelsim.ini* file; see the "[vsim] simulator control variables" (B-398)

`-multisource_delay min | max | latest`
Controls the handling of multiple PORT or INTERCONNECT constructs that terminate at the same port. Optional. By default, the Module Input Port Delay (MIPD) is set to the **max** value encountered in the SDF file. Alternatively, you may choose the **min** or **latest** of the values.

`-nocompress`
Causes VSIM to create uncompressed checkpoint files. Optional. This option must be used with the -restore option (below) to restore a simulation from an uncompressed checkpoint file.

`+no_notifier`
Timing messages will be issued for timing constraint violations, but X propagation will be prevented for these violations. Optional.

`+no_tchk_msg`
Disables timing constraint error messages. Optional.

`+notimingchecks`
Disables Verilog and VITAL timing checks for faster simulation. Optional. By default, Verilog timing check system tasks ($setup, $hold,...) in specify blocks are enabled. For VITAL, the timing check default is controlled by the ASIC or FPGA vendor, but most default to enabled.

`-quiet`
Disable 'loading' messages during batch-mode simulation. Optional.

`-restore <filename>`
Specifies that VSIM is to restore a simulation saved with the **checkpoint** command (CR-53). Optional. Use the **-nocompress** switch (above) if compression was turned off when the **checkpoint** command (CR-53) was used or if VSIM was initially invoked with **-nocompress**. See additional discussion in "How to use checkpoint/restore" (E-438); **-nocompress** is also an option of the **restore** command (CR-134).

`-sdfmin | -sdftyp | -sdfmax [<instance>=]<sdf_filename>`
Annotates VITAL or Verilog cells in the specified SDF file (a Standard Delay Format file) with minimum, typical, or maximum timing. Optional. The use of [<instance>=] with <sdf_filename> is also optional.; this is used when the back annotation it is not being done at the top level. See "Specifying SDF files for simulation" (11-282).

`-sdfnoerror`
Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue. Optional.

`-sdfnowarn`
Disables warnings from the SDF reader. Optional.

See *Chapter 4 - VHDL Simulation* for an additional discussion of SDF.

`+sdf_verbose`
Turns on the verbose mode during SDF annotation. The Main window transcript provides detailed warnings and summaries of the current annotation. Optional.

`-t [<multiplier>]<time_unit>`
Specifies the simulation time resolution. Optional; if omitted, the value specified by the Resolution (B-400) in the *modelsim.ini* file will be used. If Verilog

‘timescale directives are found, the minimum time precision will be used; <time_unit> must be one of:

fs, ps, ns, us, ms, or **sec, min,** or **hr**

The default is 1ns; the optional <multiplier> may be 1, 10 or 100. Once you’ve begun simulation, you can determine the current simulator resolution by invoking the **report** command (CR-131) with the **simulator state** option.

`-tag <string>`

Specify a string tag to append to foreign trace filenames. Optional; used with the **-trace_foreign <int>** option. Used when running multiple traces in the same directory. See "Invoking a trace" (12-334).

`-title <title>`

Specifies the title to appear for the Model*Sim* Main window. Optional. If omitted the current Model*Sim* version is the window title. Useful when running multiple simultaneous simulations. Text strings with spaces must be in quotes, i.e., "my title".

`-trace_foreign <int>`

Creates two kinds of foreign interface traces: a log of what functions were called, with the value of the arguments, and the results returned; and a set of C-language files to replay what the foreign interface side did.

The purpose of the logfile is to aid the debugging of your FLI and/or PLI code. The primary purpose of the replay facility is to send the replay file to MTI support for debugging co-simulation problems, or debugging problems for which it is impractical to send the FLI/ PLI code. See "Invoking a trace" (12-334) for more information.

`-version`

Returns the version of the simulator as used by the licensing tools, such as "Model Technology ModelSim EE/Plus vsim 5.3 DEV Simulator 1999.06 Jul 2 1999".

`-view <filename>`

Specifies the event logfile for **vsim** to read. Allows you to use VSIM to view the results from an earlier simulation. The Structure, Signals, Waveform, and List windows can be opened to look at the results stored in the logfile (other Model*Sim* windows will not open when you are viewing a logfile). See additional discussion in "Examples" (CR-218).

`-wav <filename>`

Specifies the name of the simulator event logfile to create. The default is *vsim.wlf*. Optional.

`-wavslim <size>`

Specifies a size restriction in megabytes for the event portion of the WLF logfile. Optional. The default is infinite size (0). The **<size>** must be an integer.

Note that a logfile contains an event, header, and symbol portion. The size restriction is placed on the event portion only. When Model*Sim* exits, the entire header and symbol portion of the WLF file is written. Consequently, the resulting file will be larger than the size specified with **-wavslim**.

If used in conjunction with **-wavtlim**, the **-wavslim** or -**wavtlim** value that limits the file the most will take effect.

`-wavtlim <duration>`

Specifies the duration of simulation time for WLF logfile recording. Optional. The default is infinite time (0). The **<duration>** is an integer of simulation time at the current resolution; you can optionally specify the resolution if you place curly braces around the specification. For example,

```
{5000 ns}
```

sets the duration at nanoseconds regardless of the current simulator resolution.

The time range begins at current simulation time and moves back in simulation time for the specified duration. For exmaple,

```
vsim -wavtlim 5000
```

writes at least the last 5000ns of the current simulation to the logfile (the current simulation resolution in this case is ns).

If used in conjunction with **-wavslim**, the **-wavslim** or -**wavtlim** value that limits the file the most will take effect.

Note: The **-wavslim** and **-wavtlim** switches were designed to help users limit simulation runs for very large files. When small values are used for these switches, the values may be overridden by the internal granularity limits of the WLF logfile format.

Arguments, VHDL

`-foreign <attribute>`

Specifies the foreign module to load. Optional. A particular C function from a shared library may be specified with:

```
vsim -foreign <name of C function> <path to shared library>
```

Syntax for the attribute is further described in "Declaring the FOREIGN attribute" (12-298).

`-g<Name=Value> ...`

Note there is no space between **-g** and <**Name=Value**>. Specifies a value for all generics in the design with the given name that have not received explicit values in generic maps (such as top-level generics and generics that would otherwise receive their default value). Optional.

Name is the name of the generic parameter, exactly as it appears in the VHDL source (case is ignored). **Value** is an appropriate value for the declared data type of the generic parameter.

Make sure the **Value** you specify is appropriate for the declared data type. Type mismatches will cause the specification to be ignored (including no error messages).

No spaces are allowed anywhere in the specification, except within quotes when specifying a string value. Multiple **-g** options are allowed, one for each generic parameter.

Name may be prefixed with a relative or absolute hierarchical path to select generics in an instance-specific manner. For example,

Specifying `-g/top/u1/tpd=20ns` on the command line would only affect the *tpd* generic on the */top/u1* instance, assigning it a value of 20ns.

Specifying `-gu1/tpd=20ns` affects the *tpd* generic on all instances named *u1*.

Specifying `-gtpd=20ns` affects all generics named *tpd*.

If more than one **-g** option selects a given generic the most explicit specification takes precedence. For example,

```
vsim -g/top/ram/u1/tpd_hl=10ns -gtpd_hl=15ns top
```

This command sets *tpd_hl* to 10ns for the */top/ram/u1* instance. However, all other *tpd_hl* generics on other instances will be set to 15ns.

Limitation: Composite typed generics (arrays and records) may not be set from the command line. Generics of string type may be set, however. For example,

```
-gstrgen='"This is a string"'
```

`-G<Name=Value> ...`

Note there is no space between **-G** and <**Name=Value**>. Same as **-g** except that it will also override generics that received explicit values in generic maps. Optional.

Name is the name of the generic parameter, exactly as it appears in the VHDL source (case is ignored). **Value** is an appropriate value for the declared data type of the generic parameter.

Note: Make sure the **Value** you specify is appropriate for the declared data type. Type mismatches will cause the specification to be ignored (including no error messages).

No spaces are allowed anywhere in the specification, except within quotes when specifying a string value. Multiple **-G** options are allowed, one for each generic parameter.

Name may be prefixed with a relative or absolute hierarchical path to select generics in an instance-specific manner. For example,

Specifying `-G/top/u1/tpd=20ns` on the command line would only affect the *tpd* generic on the */top/u1* instance, overriding it with a value of 20ns.

Specifying `-Gu1/tpd=20ns` affects the *tpd* generic on all instances named *u1*.

Specifying `-Gtpd=20ns` affects all generics named *tpd*.

If more than one **-G** option selects a given generic the most explicit specification takes precedence. For example,

```
vsim -G/top/ram/u1/tpd_hl=10ns -Gtpd_hl=15ns top
```

This command sets *tpd_hl* to 10ns for the */top/ram/u1* instance. However, all other *tpd_hl* generics on other instances will be set to 15ns.

Limitation: Composite typed generics (arrays and records) may not be set from the command line. Generics of string type may be set, however. For example,

```
-Gstrgen='"This is a string"'
```

`-nocollapse`
Disables the optimization of internal port map connections. Optional.

`-noglitch`
Disables VITAL glitch generation. Optional.

See *Chapter 4 - VHDL Simulation* for additional discussion of VITAL.

`+no_glitch_msg`
Suppresses VITAL glitch messages. Optional.

`-std_input <filename>`
Specifies the file to use for the VHDL TextIO STD_INPUT file. Optional

`-std_output <filename>`
Specifies the file to use for the VHDL TextIO STD_OUTPUT file. Optional

`-strictvital`
Exactly match the VITAL package ordering for messages and delta cycles. Optional. Useful for eliminating delta cycle differences caused by optimizations not addressed in the VITAL LRM. Using this will reduce simulator performance.

`-vcdread <filename>`
Simulates theVHDL top-level design from the specified VCD file. Optional. The VCD file must have been created in a previous simulation using the **vcd file** command (CR-163) with the **-nomap** and **-direction** options.

`-vital2.2b`
Select SDF mapping for VITAL 2.2b (default is VITAL 95). Optional.

Arguments, Verilog

`+alt_path_delays`
Use the current output value instead of pending value when selecting inertial specify path output delay. Optional.

`-hazards`
Enables hazard checking in Verilog modules. Optional.

`-L <library_name> ...`
Specifies the design library to search for the specified configuration, entity or module. Optional, if omitted the **work** library is used. If multiple libraries are specified, each must be preceded by the -L option.

`-Lf <library_name> ...`
Same as -L but libraries are searched before ‘uselib. Optional.

`+mindelays`
Selects minimum timing from Verilog **min:typ:max** expressions. Optional.

`+maxdelays`
Selects maximum timing from Verilog **min:typ:max** expressions. Optional.

`+no_pulse_msg`
Disables path pulse error warning messages. Optional.

`+nosdfwarn`
Disables warning from SDF annotator. Optional.

`+nosdferror`
Errors issued by the SDF annotator while loading the design prevent the simulation from continuing, whereas warnings do not. Changes SDF errors to warnings so that the simulation can continue. Optional.

`+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 the code for charge decay warnings is DECAY, so use +nowarnDECAY to disable them.

`-pli "<object list>"`
Loads a space-separated list of PLI shared objects. Optional. The list must be quoted if it contains more than one object. This is an alternative to specifying PLI objects in the Veriuser entry in the *modelsim.ini* file, see "Preference variables located in INI and MPF files" (B-394).

`+<plusarg>`
Arguments preceded with "+" are accessible by the Verilog PLI routine **mc_scan_plusargs**. Optional.

`+pulse_e/<percent>`
Sets module path pulse error limit as percentage of path delay. Optional.

`+pulse_r/<percent>`
Sets module path pulse rejection limit as percentage of path delay. Optional.

`+typdelays`
Selects typical timing from Verilog **min:typ:max** expressions. Optional. Default.

Arguments, design-unit

The following library/design-unit arguments may be used with **vsim**. If no design-unit specification is made, VSIM will open the Load a Design dialog box. Multiple design units may be specified for Verilog modules and mixed VHDL/Verilog configurations.

`<library_name>.<design_unit>`
Specifies a library and associated design unit; multiple library/design unit specifications can be made. Optional. If no library is specified, the **work** library is used.

The <design_unit> may be one of the following:

`<configuration>`
Specifies the VHDL configuration to simulate.

`<module> ...`
Specifies the name of one or more top-level Verilog modules to be simulated. Optional.

`<entity> [(<architecture>)]`
Specifies the name of the top-level entity to be simulated. Optional. The entity may have an architecture optionally specified; if omitted the last architecture compiled for the specified entity is simulated. An entity is not valid if a configuration is specified.

Note: Most UNIX shells require arguments containing () to be single-quoted to prevent special parsing by the shell. See the examples below.

Examples

`vsim -gedge="low high" -gVCC=4.75 cpu`
Invokes VSIM on the entity **cpu** and assigns values to the generic parameters **edge** and **VCC**.

`vsim -view my_design.i03`
Instructs VSIM to view the results of a previous simulation run stored in the logfile, *my_design.i03*. Use the **-wav** option to specify the name of the signal logfile to create if you plan to create many files for later viewing. For example:

```
vsim -wav my_design.i01 my_asic structure
vsim -wav my_design.i02 my_asic structure
...
```

`vsim -sdfmin /top/u1=sdf1`
Annotates instance */top/u1* using the minimum timing from the SDF file *sdf1*.

Use multiple switches to annotate multiple instances:

```
vsim -sdfmin /top/u1=sdf1 -sdfmin /top/u2=sdf2 top
```

`vsim 'mylib.top(only)' gatelib.cache_set`
This example searches the libraries for **mylib** *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.

Note: The single quotes surrounding the (); this prevents special parsing by the UNIX shell.

vsim<info>

The **vsim**<**info**> commands return information about the current VSIM executable.

`vsimDate`
Returns the date the executable was built, such as "Apr 10 1997".

`vsimId`
Returns the identifying string, such as "Model*Sim* 5.1".

`vsimVersion`
Returns the version as used by the licensing tools, such as "1997.04".

vsource

The **vsource** command displays an HDL source file in the VSIM Source window. This command is used in order to set a breakpoint in a file other than the one currently displayed in the Source window (10-200).

Syntax

```
vsource
      [<filename>]
```

Arguments

<filename>
Specifies a relative or full pathname. Optional. If filename is omitted the source file for the current design context is displayed.

Examples

```
vsource design.vhd
vsource /old/design.vhd
```

.wave.tree zoomfull

The **.wave.tree zoomfull** command redraws the display to show the entire simulation from time 0 to the current simulation time. The behavior is the same as Wave window (10-212) **Zoom > Zoom Full** menu selection.

Syntax

```
.wave.tree zoomfull
```

Arguments

None.

.wave.tree zoomrange

The **.wave.tree zoomrange** command allows you to enter the beginning and ending times for a range of time units to be displayed. The behavior is the same as Wave window (10-212) **Zoom > Zoom Range** menu selection.

Syntax

```
.wave.tree zoomrange
      f1 f2
```

Arguments

`f1 f2`

Sets the waveform display to zoom from time f1 to f2, where f1 and f2 are floating point numbers. Required.

Either range number may include an optional VHDL resolution time-unit. The resolution and range number must be enclosed in either quotes or curly brackets, see the example below. If not specified the resolution defaults to the UserTimeUnit (B-401) set in the *modelsim.ini* file.

Examples

.wave.tree zoomrange .5 {1.750 ms}

wlf2log

The **wlf2log** command translates a Model*Sim* logfile (*vsim.wlf*) to a QuickSim II logfile. The command reads the *vsim.wlf* logfile generated by the **add list**, **add wave**, or **log** commands in the simulator and converts it to the QuickSim II logfile format.

You must exit Model*Sim* before running **wlf2log** because *vsim.wlf* (or other user-specified wave files) are updated dynamically.

Syntax

```
wlf2log -inout
      [-help] [-input] [-output] [-inout] [-internal]
      [-l <instance_path>] [-4.1]
      [-4.3] [-quiet] [-o <outfile>] <wavfile>
```

Arguments

`-help`
Displays a list of command options with a brief description for each. Optional.

`-input`
Lists only the input ports. Optional. This may be combined with the -output, -inout, or -internal switches.

`-output`
Lists only the output ports. Optional. This may be combined with the -input, -inout, or -internal switches.

`-inout`
Lists only the inout ports. Optional. This may be combined with the -input, -output, or -internal switches.

`-internal`
Lists only the internal signals. Optional. This may be combined with the -input, -output, or -inout switches.

`-l <instance_path>`
Lists the signals at or below the specified HDL instance path within the design hierarchy. Optional.

`-4.1`
Reads older version (pre-4.2) .wav files. Optional.

`-4.3`
Reads intermediate version (4.2 and 4.3) .wav files. Optional.

`-quiet`
Disables error message reporting. Optional.

`-o <outfile>`
Directs the output to be written to the file specified by <outfile>. Optional. The default destination for the logfile is standard out.

`<wavfile>`
Specifies the Model*Sim* logfile that you are converting. Required.

Additional information for QuickSim II users

In some cases your original QuickHDL/Model*Sim* simulation results (in your *vsim.wav* file) may contain signal values that do not directly correspond to qsim_12state values. The resulting QuickSim II logfile generated by **wlf2log** may contain state values that are surrounded by single quotes, e.g. '0' and '1'. To make this logfile compatible with QuickSim models (that expect qsim_12state) you need to use a QuickSim II function named $convert_wdb().

This function was created to convert logfiles resulting from VHDL simulation that used std_logic and std_ulogic since these data types do not correlate to QuickSim's 12 simulation states. Other VHDL data types such as qsim_state or bit (2 state) do not require conversion as they are directly compatible with qsim_12state QuickSim II Waveform Databases (WDB).

The following procedure can be used to convert a wlf2log-generated logfile into a compatible QuickSim WDB. The procedure below shows how to convert the logfile while loaded into memory in QuickSim II.

1. Load the logfile (the output from **wlf2log**) into a WDB other than "forces". "Forces" is the default WDB, so you need to choose a unique name for the WDB when loading the logfile (for example, "fred").

2. Enter the following at the command prompt from within QuickSim:

```
$convert_wdb("fred",0)
```

The first argument, which is "fred", is the name of the new WDB to be created. The second argument, which is 0, specifies the type of conversion. At this time only one type of conversion is supported. The value 0 specifies to convert std_logic or std_ulogic into qsim_12state.

3 Do a connect_wdb (either through the pulldown menus, the "Connect WDB" pallette icon under "Stimulus", or a function invocation). You specify the name of the WDB that you originally loaded logfile into (in this case, "fred").

At this point you can issue the "run" command and the stimulus in the converted logfile will be applied. Before exiting the simulation you should save the new WDB ("fred") as a WDB or logfile so that it can be loaded again in the future. The new WDB or logfile will contain the correct qsim_12state values eliminating the need to re-use convert_wdb().

when

The **when** command allows you to instruct VSIM to perform actions when the specified conditions are met. For example, you can use the **when** command to break on a signal value or at a specific simulator time (see "Time-based breakpoints" (CR-229)) . Conditions can include the following HDL items: VHDL signals, and Verilog nets and registers. Use the **nowhen** command (CR-105) to deactivate **when** commands.

The **when** command uses a when_condition_expression to determine whether or not to perform the action. The when_condition_expression uses a simple restricted language (that is not related to Tcl), which permits only four operators and operands that may be either HDL item names, signame'event, or constants. VSIM evaluates the condition every time any item in the condition changes, hence the restrictions.

With no arguments, **when** will list the currently active when statements and their labels (explicit or implicit).

Syntax

```
when
      [[-label <label>]
      {<when_condition_expression>} {<command>}]
```

Arguments

`-label <label>`
Used to identify individual **when** commands. Optional.

`{<when_condition_expression>}`
Specifies the conditions to be met for the specified <command> to be executed. Required. The condition is evaluated in the simulator kernal and can be an item name, in which case the curly braces can be omitted. The command will be executed when the item changes value. The condition can be an expression with these operators:

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

Name	Operator
OR	||, OR

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

The formal BNF syntax is:

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

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

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

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

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

`{<command>}`

The command(s) for this argument are evaluated by the Tcl interpreter within the Model*Sim* GUI. Any VSIM or Tcl command or series of commands are valid with one exception, the **run** command (CR-139) may not be used with the **when** command. Required. The command sequence usually contains the **stop** command (CR-152) that sets a flag to break the simulation run after the command sequence is completed. Multiple-line commands can be used.

Note: If you want to stop the simulation using a **when** command, you must use a **stop** command (CR-152) within your when statement. DO NOT use an **exit** command (CR-84) or a **quit** command. The **stop** command acts like a breakpoint at the time it is evaluated. See "Ending the simulation with the stop command" (CR-228) for an example.

Examples

The **when** command below instructs the simulator to display the value of item c in binary format when there is a clock event, the clock is 1, and the value of b is 01100111, and then 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 is labeled "a" and will cause VSIM to echo the message "b changed" whenever the value of the item b changes.

```
when -label a b {echo "b changed"}
```

The multi-line **when** command below does not use a label and has two conditions. When the conditions are met, an **echo** (CR-75) and a **stop** (CR-152) command will be executed.

```
when {b = 1
     and c /= 0 } {
     echo "b is 1 and c is not 0"
     stop
}
```

Batch mode simulations (see How to use checkpoint/restore (E-438)) are often structured as "run until condition X is true," rather than "run for X time" simulations. The multi-line **when** command below sets a done condition and executes an **echo** (CR-75) and a **stop** (CR-152) command when the condition is reached.

Ending the simulation with the stop command

The simulation will not stop (even if a **quit -f** command is used) unless a **stop** command is executed. To exit the simulation and quit Model*Sim*, use an approach like the following:

```
when {/done_condition == '1'} {echo "End condition reached"
        if [bathc_mode] {
            set DoneConditionReached 1
            stop }
}
run 1000 us
if {$DoneConditionReached == 1} {
        quit -f
}
```

Time-based breakpoints

You can build time-based breakpoints into a **when** statement with the following syntax.

For absolute time (indicated by @) use:

```
when {$now = @1750ns} {stop}
```

You can also use:

```
when {errorFlag = '1' OR $now = 2ms} {stop}
```

This example adds 2ms to simulation time at which the **when** statement is first evaluated, then stops.

where

The **where** command displays information about the system environment. This command is useful for debugging problems where VSIM cannot find the required libraries or support files.

Syntax

```
where
```

Arguments

None.

Description

The **where** command displays three important system settings:

`current directory`
: This is the current directory that VSIM was invoked from, or was specified on the VSIM command line. Once in VSIM the current directory cannot be changed.

`current project file`
: This is the initialization file VSIM is using. All library mappings, are taken from here. Window positions, and other parameters are taken from the *modelsim.tcl* file.

`work library`
: This is the library that VSIM used to find the current design unit that is being simulated.

write format

The **write format** command records the names and display options of the HDL items currently being displayed in the List or Wave window. The file created is comprised of a file of **add list** (CR-22), **add wave** (CR-33), and **configure** (CR-54) commands. This file may be invoked with the **do** command (CR-67) to recreate the List or Wave window format on a subsequent simulation run.

Syntax

```
write format
      list | wave <filename>
```

Arguments

`list | wave`
Specifies that the contents of either the List or the Wave window are to be recorded. Required.

`<filename>`
Specifies the name of the output file where the data is to be written. Required.

Examples

```
write format list alu_list.do
```
Saves the current data in the List window in a file named *alu_list*.do.

```
write format wave alu_wave.do
```
Saves the current data in the Wave window in a file named *alu_wave.do*.

See also

add list command (CR-22), and the **add wave** command (CR-33)

write list

The **write list** command records the contents of the most recently opened, or specified List window in a list output file. This file contains simulation data for all HDL items displayed in the List window: VHDL signals - and Verilog nets and registers.

Syntax

```
write list
      [-events] [-window <wname>] <filename>
```

Arguments

`-events`
Specifies to write print-on-change format. Optional. Default is tabular format.

`-window <wname>`
Specifies a List window using the full Tk path. Optional. Default is to use the List window set by the **view** command (CR-180).

`<filename>`
Specifies the name of the output file where the data is to be written. Required.

Examples

`write list alu.lst`
Saves the current data in the default List window in a file named *alu.lst*.

`write list -win .list1 group1.list`
Saves the current data in window 'list1' in a file named *group1.list*.

See also

write tssi command (CR-236)

write preferences

The **write preferences** command saves the current GUI preference settings to a Tcl preference file. Settings saved include the current window location and size.

Syntax

```
write preferences
      <preference file name>
```

Arguments

`<preference file name>`
Specify the name for the preference file. Optional. If the file is named *modelsim.tcl*, Model*Sim* 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 (B-392) environment variable.

See also

You can modify variables by editing the preference file with the Model*Sim* **notepad** (CR-104):

```
notepad <preference file name>
```

write report

The **write report** command prints a summary of the design being simulated including a list of all design units (VHDL configurations, entities, and packages and Verilog modules) with the names of their source files.

Syntax

```
write report
      [<filename>]
```

Arguments

`<filename>`
Specifies the name of the output file where the data is to be written. Optional. If the <filename> is omitted, the report is written to the Transcript window.

Examples

`write report alu.rep`
Saves information about the current design in a file named *alu.rep*.

write transcript

The **write transcript** command writes the contents of the Main window transcript to the specified file. The resulting file can be used to replay the transcribed commands as a DO file (macro).

Syntax

```
write transcript
      [<filename>]
```

Arguments

`<filename>`
Specifies the name of the output file where the data is to be written. Optional. If the <filename> is omitted, the transcript is written to a file named *transcript*. The size of this file can be controlled with the MTI_TF_LIMIT (B-391).

See also

do command (CR-67)

write tssi

The **write tssi** command records the contents of the default or specified List window in a "TSSI format" file. The file contains simulation data for all HDL items displayed in the List window that can be converted to TSSI format (VHDL signals and Verilog nets). A signal definition file is also generated.

The List window needs to be using symbolic radix in order for **write tssi** to produce useful output.

Syntax

```
write tssi
      [-window <wname>] <filename>
```

Arguments

`-window <wname>`
Specifies a List window using the full Tk path. Optional. Default is to use the List window set by the **view** command (CR-180).

`<filename>`
Specifies the name of the output file where the data is to be written. Required.

Description

"TSSI format" is documented in the Summit Design document *ASCII I/O Interface*, dated August 31, 1992. In that document, it is called Standard Events Format (SEF).

If the <filename> has a file extension (e.g., *listfile.lst*), then the definition file is given the same file name with the extension .def (e.g., *listfile.def*). The values in the listfile are produced in the same order that they appear in the list window. The directionality is determined from the port type if the item is a port, otherwise it is assumed to be bidirectional (mode INOUT).

Items that can be converted to SEF are VHDL enumerations with 255 or fewer elements and Verilog nets. The enumeration values U, X, 0, 1, Z, W, L, H and - (the enumeration values defined in the IEEE Standard 1164 **std_ulogic** enumeration) are converted to SEF values according to the table below. Other values are converted to a question mark (?) and cause an error message. Though the WRITE TSSI command was developed for use with **std_ulogic**, any signal which uses only the values defined for **std_ulogic** (including the VHDL standard type **bit**) will be converted.

std_ulogic State Characters	SEF State Characters		
	Input	Output	Bidirectional
U	N	X	?
X	N	X	?
0	D	L	0
1	U	H	1
Z	Z	T	F
W	N	X	?
L	D	L	0
H	U	H	1
-	N	X	?

Bidirectional logic values are not converted because only the resolved value is available. The TSSI 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 TSSI ASCII In Converter and ASCII Out Converter are part of the TSSI software from Summit Design. Model*Sim* outputs a vector file, and Summit's tools determine whether the bidirectional signals are driving or not.

See also

tssi2mti command (CR-159)

write wave

The **write wave** command records the contents of the most currently opened, or specified Wave window in PostScript format. The output file can then be printed on a PostScript printer.

Syntax

```
write wave
      [-window <wname>] [-width <real_num>] [-height <real_num>]
      [-margin <real_num>] [-start <time>] [-end <time>] [-perpage
      <time>]
      [-pagecount <n>] [-landscape] [-portrait] <filename>
```

Arguments

`-window <wname>`
Specifies a Wave window using the full Tcl path. Optional. Default is to use the Wave window set by the **view** command (CR-180).

`-width <real_num>`
Specifies the paper width in inches. Optional. Default is 8.5.

`-height <real_num>`
Specifies the paper height in inches. Optional. Default is 11.0.

`-margin <real_num>`
Specifies the margin in inches. Optional. Default is 0.5.

`-start <time>`
Specifies the start time (on the waveform time scale) to be written. Optional.

`-end <time>`
Specifies the end time (on the waveform time scale) to be written. Optional.

`-perpage <time>`
Specifies the time width per page of output. Optional.

`-pagecount <n>`
Specifies the number of pages to use horizontally along the time axis. Optional.

`-landscape`
Use landscape (horizontal) orientation. Optional. This is the default orientation.

`-portrait`
Use portrait (vertical) orientation. Optional. The default is landscape (horizontal).

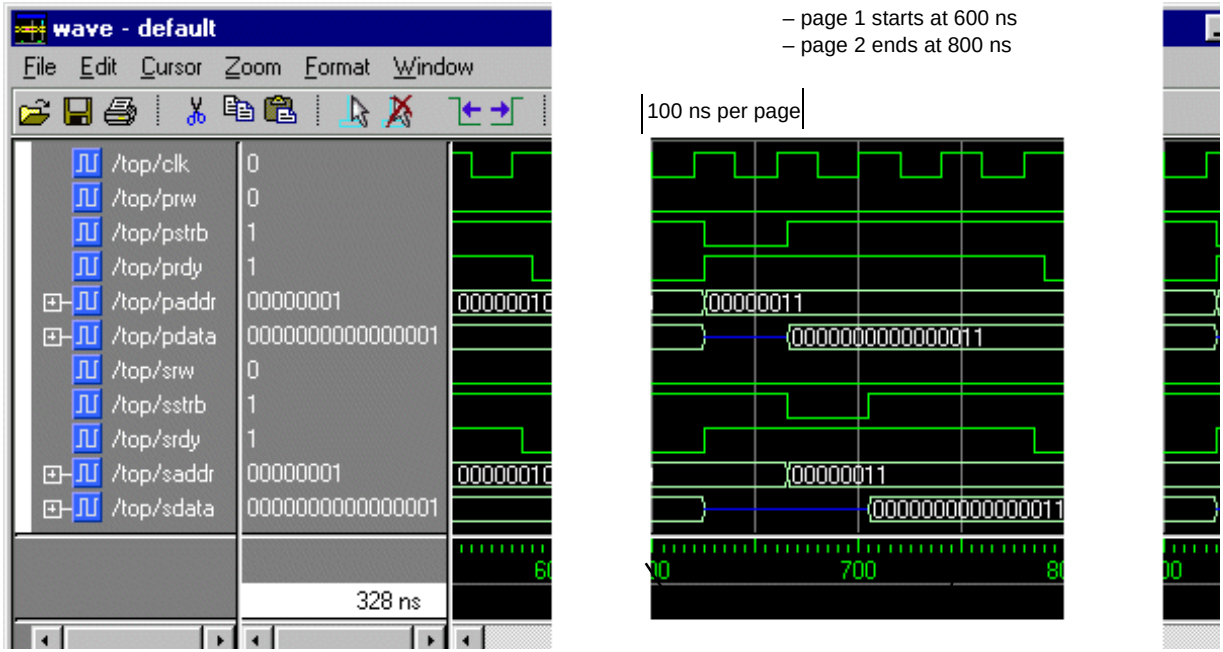
`<filename>`
Specifies the name of the PostScript output file. Required.

Examples

`write wave alu.ps`
Saves the current data in the Wave window in a file named *alu.ps*.

`write wave -win .wave2 group2.ps`
Saves the current data in window ‘wave2’ in a file named *group2.ps*.

`write wave -start 600ns -end 800ns -perpage 100ns top.ps`
Writes two separate pages to *top.ps* as indicated in the illustration (the actual PostScript print out will show all items listed in the Wave window, not just the portion in view):



To make the job of creating a PostScript waveform output file easier, use the **File > Write Postscript** menu selection in the Wave window. See "Printing and saving waveforms" (10-238) for more information.

Command Syntax and Conventions

Chapter contents

This chapter documents syntax for all Model*Sim* commands. Documentation for Tcl commands, and FlexLM commands are available in these locations:

- **Tcl man pages**
  available in HTML format - use this Main window menu selection:
  Help > Tcl man pages
- **FLEXlm commands**
  for tools available within Model*Sim* see "License administration tools" (D-434) in the User's Manual; the complete FLEXlm user's manuals is available at:
  http://www.globetrotter.com/manual.htm

Syntax conventions

These syntax elements are used within all Model*Sim* command documentation:

Syntax notation	Description
< >	angled brackets surrounding a syntax item indicate a user-defined argument; do not enter the brackets in commands
[ ]	square brackets indicate an optional item; if the brackets surround several words, all must be entered as a group; the brackets are not 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 examples
#	comments included with commands are preceded by the number sign (#); useful for adding comments to DO files (macros)

Comments in argument files loaded with -f <filename>

Argument files may be loaded with the **-f <filename>** argument of the **vcom**, **vlog**, and **vsim** commands. The **-f <filename>** argument specifies a file that contains more command line arguments. It allows complex arguments to be reused without retyping.

Comments within the argument files follow these rules:

- All text in a line beginning with // to its end is treated as a comment.
- All text bracketed by /* ... */ is treated as a comment.

Also, program arguments can be placed on separate lines in the argument file, with the newline characters treated as space characters. There is no need to put '\' at the end of each line.

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

Command return values

All simulator commands are invoked using Tcl. For most commands that write information to the Main window, that information is also available as a Tcl result. By using command substitution the results can be made available to another command or assigned to a Tcl variable. For example:

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

sets variable "aluinputs" to the result of the **find** command (CR-85).

Command shortcuts

You may abbreviate command syntax, but there's a catch. The minimum 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 Model*Sim* does not allow command name abbreviations in macro files. This minimizes your need to maintain macro files as new commands are added.

Command history shortcuts

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

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

Numbering conventions

Numbers in Model*Sim* /PLUS can be expressed in either VHDL or Verilog style. Two styles can be used for VHDL numbers, one for Verilog.

VHDL numbering conventions

The first of two VHDL number styles is:

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

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

Examples

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

The second VHDL number style is:

```
base "value"
```

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

Examples

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

Verilog numbering conventions

Verilog numbers are expressed in the style:

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

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

Examples

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

HDL item pathnames

VHDL and Verilog items are organized hierarchically. Each of the following HDL items creates a new level in the hierarchy:

- **VHDL**
  component instantiation statement, block statement, and package
- **Verilog**
  module instantiation, named fork, named begin, task and function

Multiple levels in a pathname

Multiple levels in a pathname are separated by the character specified in the PathSeparator variable. The default is "/", but it can be set to any single character, such as "." for Verilog naming conventions, or ":" for VHDL IEEE 1076-1993 naming conventions. See the PathSeparator (B-400) for more information.

Absolute path names

Absolute path names begin with the path separator character. The first name in the path should be the name of a top-level entity or module, but if you leave it off then the first top-level entity or module will be assumed. VHDL designs only have one top-level, so it doesn't matter if it is included in the pathname. For example, if you are referring to the signal CLK in the top-level entity named top, then both of the following pathnames are correct:

```
/top/clk
/clk
```

Note: Since Verilog designs may contain multiple top-level modules, a path name may be ambiguous if you leave off the top-level module name.

Relative pathnames

Relative pathnames do not start with the path separator, and are relative to the current environment. The current environment defaults to the first top-level entity or module and may be changed by the environment command or by clicking on hierarchy levels in the structure window. Each new level in the pathname is first searched downwards relative to the current environment, but if not found is then searched for upwards (same search rules used in Verilog hierarchical names).

Indexing signals, memories and nets

VHDL array signals, and Verilog memories and vector nets can be sliced or indexed. Indexes must be numeric, since the simulator does not know the actual index types. Slice ranges may be represented in either VHDL or partial Verilog syntax, irrespective of the setting of the PathSeparator (B-400). The syntax is *partial* Verilog because the range must be contained within parentheses and not in Verilog square brackets. For example,

```
mysignal(31:0)
```

specifies a slice of an array item in partial Verilog syntax.

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. In contrast, all Verilog names are case sensitive.

Names in VSIM 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 trailing space.  Compatible with 5.2

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

Naming fields in VHDL signals

Fields in VHDL record signals can be specified using the form:

```
signal_name.field_name
```

Examples:

Syntax	Description
clk	specifies the item clk in the current environment
/top/clk	specifies the item clk in the top-level design unit.
/top/block1/u2/clk	specifies the item clk, two levels down from the top-level design unit
block1/u2/clk	specifies the item clk, two levels down from the current environment
array_sig(4)	specifies an index of an array item
array_sig(1 to 10)	specifies a slice of an array item in VHDL syntax
mysignal(31:0)	specifies a slice of an array item in partial Verilog syntax
record_sig.field	specifies a field of a record

Wildcard characters

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

Syntax	Description
*	matches any sequence of characters
?	matches any single character

You can use square brackets [ ] in wildcard specifications if you place the entire name in curly braces { }:

Syntax	Description
{[abcd]}	matches any character in the specified set
{[a-d]}	matches any character in the specified range
{[^a-d]}	matches any character not in the set

Examples

```
*
```
Matches all items.

```
{*[0-9]}
```
Matches all items ending in a digit.

```
{?in*[0-9]}
```
Matches such item names as pin1, fin9, and binary2.

Model*Sim* variables

Several variables are available to control simulation, provide simulator state feedback, or modify the appearance of the Model*Sim* GUI. To take effect, some variables, such as environment variables, must be set prior to simulation.

Model*Sim* variables can be referenced in simulator commands by preceding the name of the variable with the dollar sign ($) character. The Model*Sim* uses global Tcl variables for simulator state variables, simulator control variables, simulator

preference variables and user-defined variables. Note that case is significant in variable names.

See *Chapter B - ModelSim Variables* for a complete listing of Model*Sim* variables.

Variable settings report

The **report** command (CR-131) 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:

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

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

Unless you specify otherwise as in the examples above, simulation time is always expressed using the resolution units that are specified by the UserTimeUnit variable. See the UserTimeUnit (B-401).

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.

GUI_expression_format

The GUI_expression_format is an option of several simulator commands that operate within Model*Sim*'s GUI environment. The expressions allow the use of criteria to locate and examine HDL items within the List and Wave windows. The commands that use the expression format are:

- **down | up** (CR-70)
  search for the expression in the List window
- **examine** (CR-81)
  examine the value and compute mathematical functions of items in the List window
- **right | left** (CR-136)
  search for the expression in the Wave window
- **virtual signal** (CR-193))
  define a virtual signal (a combination of real signals defined by the expression)
- **virtual function** (CR-184))
  to define a virtual function (displays as a new signal defined by the expression)
- **searchLog** (CR-144)
  search the current log files (WLF files) for the defined expression

Expressions may be typed directly on the VSIM command line, or you can use the "The GUI Expression Builder" (10-272).

Expression typing

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

Scalar types

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

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

Signal and sub-element naming conventions

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

Concatenation of signals and/or sub-elements

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 "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 type, a VHDL-style record will be created. The record object can be expanded in the signals and wave window just like an array of compatible type elements.

Concatenation syntax for VHDL

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

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

Concatenation syntax for Verilog

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

The 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 descending index range from (n-1) to 0, where n is the number of elements in the array. The "range" directive completely specifies the index range. The "ascending" directive specifies that the index start at zero and increment upwards, and the "descending" directive specifies (n-1) downto 0.

Concatenation directives

```
(range [31:0])<concatenationExpr> # Verilog syntax
(range (31 downto 0))<concatenationExpr> # VHDL syntax
(ascending) <concatenationExpr>
(descending)<concatenationExpr>
(flatten)<concatenationExpr> # no hierarchy
```

Examples in Verilog and VHDL syntax:

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

VHDL record field support

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)
```

Grouping and precedence

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

Saving expressions

Expressions created in the expression builder dialog box can be saved for future use by pressing the **SAVE** button. You will be prompted for the name of a global-level Tcl variable, to be assigned the expression string. After entering the variable name, the expression will be saved. Regardless of whether you enter a Tcl variable name or not, the expression will also be saved in the entry box drop-down cache. A previous entry in the cache can be selected by simply clicking with the mouse.

Expression syntax

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

A GUI expression can include the following elements:

Tcl macros

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

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

Constants

Type	Values
boolean value	0 1 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' 'H' 'L' 'W' '-' 1'b0 1'b1

Array constants, expressed in any of the following formats

Type	Values
VHDL # notation	<int>#<alphanum>[#] Example: 16#abc123#
VHDL bitstring	"(U|X|0|1|Z|L|H|W|-)*" Example: "11010X11"
VLOG 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.)

Variables

Variable	Type
Name of a signal	The name may be a simple name, a VHDL or VLOG style extended identifier, or a VHDL or VLOG 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 -- VLOG net -- VLOG register -- VLOG integer -- VLOG real.
NOW	Returns the value of time at the current location in the logfile as the logfile is being scanned (not the most recent simulation time).

Array variables

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

Signal attributes

<name>'event
<name>'rising
<name>'falling

Operators

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

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

Note: Arithmetic operators use the std_logic_arith package.

Casting

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

Examples

```
/top/bus and $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 1 when signal clk changes and signal /top/u3/addr is equal to hex ffae; otherwise is 0.

```
clk'rising && (mystate == reading) && (/top/u3/addr == 32'habcd1234)
```
Evaluates to a boolean 1 when signal clk just changed from low to high and signal mystate is the enumeration reading and signal /top/u3/addr is equal to the specified 32-bit hex constant; otherwise is 0.

```
(/top/u3/addr and 32'hff000000) == 32'hac000000
```
Evaluates to a boolean 1 when the upper 8 bits of the 32-bit signal /top/u3/adder equals hex ac.

```
((NOW > 23 us) && (NOW < 54 us)) && clk'rising && (mode == writing)
```
Evaluates to a boolean 1 when logfile time is between 23 and 54 microseconds, and clock just changed from low to high and signal mode is enumeration writing.

Index

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

V

W

Thank you for purchasing ModelSim!

www.model.com