VCS®/VCSi™
Quick Reference

Version C-2009.06
June 2009

Comments?
E-mail your comments about VCS documentation to
vcs_support@synopsys.com

Copyright Notice and Proprietary Information

Right to Copy Documentation

The license agreement with Synopsys permits licensee to make copies of the documentation for its internal use only. Each copy shall include all copyrights, trademarks, service marks, and proprietary rights notices, if any. Licensee must assign sequential numbers to all copies. These copies shall contain the following legend on the cover page:

> "This document is duplicated with the permission of Synopsys, Inc., for the exclusive use of _____ and its employees. This is copy number _____."

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Registered Trademarks (®)

Synopsys, AMPS, Cadabra, CATS, CRITIC, CSim, Design Compiler, DesignPower, DesignWare, EPIC, Formality, HSIM, HSPICE, iN-Phase, in-Sync, Leda, MAST, ModelTools, NanoSim, OpenVera, PathMill, Photolynx, Physical Compiler, PrimeTime, SiVL, SNUG, SolvNet, System Compiler, TetraMAX, VCS, and Vera are registered trademarks of Synopsys, Inc.

Trademarks (™)

Active Parasitics, AFGen, Apollo, Astro, Astro-Rail, Astro-Xtalk, Aurora, AvanTestchip, AvanWaves, BOA, BRT, ChipPlanner, Circuit Analysis, Columbia, Columbia-CE, Comet 3D, Cosmos, CosmosEnterprise, CosmosLE, CosmosScope, CosmosSE, Cyclelink, DC Expert, DC Professional, DC Ultra, Design Advisor, Design Analyzer, Design Vision, DesignerHDL, DesignTime, Direct RTL, Direct Silicon Access, Discovery, Dynamic-Macromodeling, Dynamic Model Switcher, EDAnavigator, Encore, Encore PQ, Evaccess,

Document Order Number hard copy not available
VCS/VCSi Quick Reference, Version A-2008.03 Beta

Table of Contents

About VCS

VCS™ (Verilog Compiled Simulator) is the complete Verilog simulator from Synopsys. VCS simulates IEEE Std 1364. VCS has additional built in system tasks and functions and compiler directives not found in the Std 1364.
VCS has a Command Language Interface (CLI) for entering commands during simulation.

Reserved Keywords

```
always          for             output          supply0
and             force           parameter       supply1
assign          forever         pmos            table
begin           fork            posedge         task
buf             function        primitive       time
bufif0          highz0          pull0           tran
bufif1          highz1          pull1           tranif0
case            if              pulldown        tranif1
casex           ifnone          pullup          tri
casez           initial         rcmos           tri0
cmos            inout           real            tri1
deassign        input           reg             triand
default         integer         release         trior
defparam        join            repeat          trireg
disable         large           rnmos           vectored
edge            macromodule     rpmos           wait
else            medium          rtran           wand
end             module          rtranif0        weak0
endcase         nand            rtranif1        weak1
endfunction     negedge         scalared        while
endmodule       nmos            small           wire
endprimitive    nor             specify         wor
endspecify      not             specparam       xnor
endtable        notif0          strength        xor
endtask         notif1          strong0
event           or              strong1
```

Concurrency

The following Verilog HDL constructs are independent processes that are evaluated concurrently in simulation time:

- module instances
- primitive instances
- procedural blocks
- continuous assignments

Lexical Conventions

The Verilog HDL is a stream of the following lexical tokens:

Token	Description
white space	blanks, tabs, new lines (carriage return), and form feeds
Comment	`//` begins a single line comment, terminated by a new line `/*` begins a multi-line comment, terminated by a `*/`

Token	Description
Operator	See “Operators” on page 19
Number	See “Integers” on page 8 and “Real numbers” on page 8
String	A sequence of characters enclosed in quotation marks, contained on a single line
Identifier	See“Identifiers (Names)” on page 7
Keyword	See “Reserved Keywords” on page 6

Identifiers (Names)

- Must begin with alphabetic or underscore characters **a-z, A-Z, _**
- Can also contain the characters **0-9** and **$**
- May use any character by escaping with a back slash (\) at the beginning of the identifier, and terminating with a white space

Examples	Notes
`adder`	Valid identifier name
`XOR`	Uppercase identifier differs from the `xor` keyword
`\reset*`	An escaped identifier (must be followed by a white space)

Logic Values

The Verilog HDL has four logic values.

Logic Value	Description
`0`	Zero, low, or false
`1`	One, high, or true
`z` or `Z`	High impedance, tri-stated, or floating)
`x` or `X`	Unknown or uninitialized (can be 0, 1 or Z)

Logic Strengths

The Verilog HDL has 8 logic strengths: 4 driving, 3 capacitive, and high impedance (no strength).

Strength Level	Strength Name	Specification Keyword	Display Mnemonic
7	Supply Drive	`supply0` `supply1`	`Su0` `Su1`
6	Strong Drive	`strong0` `strong1`	`St0` `St1`
5	Pull Drive	`pull0` `pull1`	`Pu0` `Pu1`
4	Large Capacitive	`large`	`La0` `La1`
3	Weak Drive	`weak0` `weak1`	`We0` `We1`
2	Med. Capacitive	`medium`	`Me0` `Me1`

Strength Level	Strength Name	Specification Keyword	Display Mnemonic
1	Small Capacitive	`small`	`Sm0 Sm1`
0	High Impedance	`highz0 highz1`	`HiZ0 HiZ1`

Integers

value — Unsized decimal integer

<size><base><value> — Sized integer in a specific radix (base). Where *size* is the number of bits, and *base* is the radix.

Base	Symbol	Legal Values
binary	`b` or `B`	0,1,x,X,z,z,Z, ?, _
octal	`o` or `O`	0-7, x, X, z, Z, ?, _
decimal	`d` or `D`	0-9
hexadecimal	`h` or `H`	0-9, a-f, A-F, x, X, z, Z, ?_

When assigning integer values:

- The VCS standard for unsized integers is 32-bits.
- The ? represents “don’t care.”
- Underscore are ignored (used for readability).
- Verilog always assigns values from right (lsb) to left (msb).
- When size is less than value, the upper bits are truncated.
- When size is larger than value, and the left-most bit of value is 0 or 1, zeros are left-extended to fill the size.
- When size is larger than value, and the left-most bit of value is Z or X, the Z or X is left-extended to fill the size.

Examples	Size	Base	Binary Equivalent
`10`	32 bits	decimal	`0...01010` (32-bits)
`1’b1`	1 bit	binary	`1`
`8’Haa`	8 bits	hex	`01010101`
`6’hF0`	6 bits	hex	`110000` (truncated)
`6’hF`	6 bits	hex	`001111` (zero filled)
`8’bz`	8 bits	binary	`zzzzzzzz` (z filled)

Real numbers

value.value — A real number in decimal notation.

<base><E><exponent> — A real number in scientific notation (The letter `E` is not case sensitive).

When assigning real number values:

- Real numbers are limited to the values 0-9 and underscore.
- There must be a value on either side of the decimal point.

- The exponent must be an integer.

Examples	Notes
`1.2`	Decimal notation
`0.5`	Must have value on both sides of decimal point
`3e4`	3 times 10^4 (30000)
`5.8E-3`	5.8 times 10^{-3} (0.0058)

Module Definitions

Modules are the building blocks of Verilog models. The constructs in modules are as follows:

```
module module_name (port_name, port_name, ...);
  module_port_declarations                    (See page 9)
  data_type_declarations                      (See page 10)
  primitive_instances                         (See page 12)
  module_instances                            (See page 13)
  procedural_blocks                           (See page 14)
  continuous_assignments                      (See page 21)
  task_and_function_definitions               (See page 22)
  specify_blocks                              (See page 23)
endmodule
```

- A module contains declarations, functionality, and timing.
- Module functionality may be behavioral (procedural blocks with programming statements), structural (a netlist of components), or a combination of behavioral and structural.
- Modules can instantiate other modules.
- Module constructs can be in any order, but ports and data types must be declared before they are used.

Module Port Declarations

Scalar (1-bit) port declaration syntax:

```
port_direction port_name, ...;
```

Vector (multi-bit) port declaration syntax:

```
port_direction [port_size] port_name, ,...;
```

`port_direction` is `input`, `output` or `inout` (bidirectional). *`port_size`* is a range from `[msb:lsb]` (Most-Significant-Bit to Least-Significant-Bit).

- The *`msb`* and *`lsb`* must be literal integers or parameters.
- Either little-endian or big-endian conventions may be used.
- The VCS limit on port sizes is 1 million bits.

Examples	Notes
`input a, b, sel;`	3 scalar ports
`output [7:0] result;`	Little endian convention
`inout [0:15] data_bus;`	Big endian convention

Examples	Notes
`input [15:12] addr;`	msb:lsb may be any integer
`parameter word = 32;` `input [word - 1:0] addr;`	Constants and expressions can be used

Data Type Declarations

Every signal or variable must have a data type associated with it. There are two classes of data types, nets and registers.

Net Data Types

Net data types connect components together. Use a net data type when:

- A signal is driven by the output of some device.
- A signal is also declared as an input port or inout port.
- A signal is on the left-hand side of a continuous assignment.

Net Data Type	Functionality
`wire tri`	Basic interconnecting wire
`wor trior`	Wired outputs OR together
`wand triand`	Wired outputs AND together
`tri0 tri1`	Pulls down or up when not driven
`supply0 supply1`	Constant logic 0 or 1 (supply strength)
`trireg`	Stores last value when tri-stated (capacitance strength)

Implicit Declarations

Signals included in module instance port connection lists and primitive instance terminal connection lists, but are not explicitly declared, are implicitly declared to be scalar wire nets. You can change this default with the `` `default_nettype `` compiler directive.

Register Data Types

Register data types are used as variables in procedural blocks. Use a register data type when the signal is on the left-hand side of a procedural assignment.

Register Type	Functionality
`reg`	Unsigned variable of any bit size
`integer`	Signed 32-bit variable
`time`	Unsigned 64-bit variable
`real`	Double-precision floating point variable
`realtime`	Another double-precision floating point variable Functionally the same as real.

Other Data Types

Special data types that are neither nets nor registers:

Other Types	Functionality
`parameter`	Run-time constant for storing integers, real numbers, time, delays, or ASCII strings. Parameters in a module may be redefined for each instance of the module (see "Overriding Module Parameter Values" on page 14).
`specparam`	Specify block constant for storing integers, real numbers, time, delays or ASCII strings.
`event`	A momentary flag with no logic value or data storage. Often used for synchronizing concurrent activities within a module.

Data Type Declaration Syntax

Data Type Declaration Syntax:
net_type *[size] #delay net_name, net_name, ...;*
net_type *(drive_strength) [size] #delay net_name = continuous_assignment;*
trireg *(capacitance_strength) [size] #delay net_name, net_name,...;*
register_type *[size] register_name, register_name,...;*
register_type *[size] array_name [array_size];*
parameter *constant_name = value, constant_name = value, ...;*
specparam *constant_name = value, constant_name = value, ...;*
event *event_name, event_name, ...;*

You can only specify `#delay` on net data types. The syntax for delays is the same as when specifying delays on primitives. See "Primitive Delays" on page 13.

`size` is a range from `[msb:lsb]` (Most-Significant-Bit to Least-Significant-Bit).

- The `msb` and `lsb` must be positive integers, integer parameters or an expression that resolves to an integer.
- Either little-endian or big-endian conventions may be used.
- The VCS limit on vector size is 1 million bits.

`array_size` is `[first_address:last_address]`.

- `first_address` and `last_address` must be positive integers, integer parameters, or an expression that resolves to integer.
- Either ascending or descending address order can be used.
- The VCS limit on the address range is 2^{32}-1.

drive_strength (optional) is specified as `(strength0, strength1)` or `(strength1, strength0)`. See "Logic Strengths" on page 7 for strength keywords.

Data Type Examples	Notes
`wire a, b, c;`	3 scalar nets
`tri1 [7:0] data_bus;`	8-bit net, pulls-up when tri-stated
`reg [1:8] result1, result2;`	Two 8-bit unsigned variables
`reg [7:0] RAM [0:1023];`	A variable array that is 8-bits wide, with 1K of addresses
`wire #(2.4,1.8) carry_bit;`	A net with rise and fall delays
`wire (strong1, pull0) sum = a+b;`	A net with drive strengths and a continuous assignment
`trireg (large) ram_cell;`	A net with large capacitance

Primitive Instances

primitive_type (drive_strength) #delay instance_name (terminal, terminal, ...);

Primitive Types

Primitive Type		Terminal Order and Quantity

`and`
`or`
`xor`
	`nand`
	`nor`
	`xnor`
		(1-output, 1-or-more-inputs)

`buf`	`not`	(1-or-more-outputs, 1-input)
`bufif0` `bufif1`	`notif0` `notif1`	(1-output, 1-input, 1-control-input)
`pmos` `rpmos`	`nmos` `rnmos`	(pmos, rpmos 0 enabled) (nmos, rnmos, rnmos 1 enabled)
`cmos`	`rcmos`	(1-output, 1-input, n-cntrl, p-cntrl)
`tran`	`rtran`	(2-bidirectional-inouts)
`tranif0` `rtranif0`	`tranif1` `rtranif1`	(2-bidirectional-inouts, 1-control)
`pullup`	`pulldown`	(1-output)
User Defined Primitive		(1-output, 1-or-more-inputs)

Primitive Drive Strengths

drive_strength (optional) is specified as `(strength0, strength1)` or `(strength1, strength0)`. The default strength is `(strong1, strong0)`. See "Logic Strengths" on page 7.

Only gate and user defined primitives may specify drive strengths. Switch primitives (`tran`, etc.) pass the input strength to the output. Resistive switches (`rtran`, etc.) reduce the strength as it is passed through.

Primitive Delays

- Separate rise, fall, and turn-off delays may be specified.
- Minimum, typical, and maximum delays may be specified.
- Integers and real numbers may be used
- If no delay is specified, the default is zero delay

Delay Syntax
`#delay` or `#(delay)` Single delay for all output transitions
`#(delay, delay)` Separate delays for rising and falling transitions
`#(delay, delay, delay)` Separate delays for rising, falling, and turn-off transitions
`#(min:typ:max)` Minimum, typical, and maximum delays for all transitions
`#(min:typ:max,min:typ:max)` Minimum, typical, and maximum delays for rising and falling transitions
`#(min:typ:max,min:typ:max,min:typ:max)` Minimum, typical, and maximum delays for rising, falling and turn-off transitions

Primitive Instance Name

Primitive instance names are optional.

Module Instances

When you instantiate a module in another module you specify the signals that connect to the module ports. You can connect signals to ports by ordered list or specify connections by name.
You can override parameter values in the instantiated module.

Port Order Instantiation

```
module_name instance_name (signal, signal, ...);
```

Lists the signals connected to the module in the same order as the port list in the module header. An unconnected port is designated by two commas with no signal between them.

Port Name Instantiation

```
module_name instance_name (.port_name(signal),
.port_name(signal), ...);
```

Lists the port name and signal connected to it, in any order.

Overriding Module Parameter Values

When you instantiate a module you can pass different values to the parameters inside that module. You can specify different values between the module name and the instance name with something that looks like a delay specification. List values in the order of parameter declarations in the module:

```
module_name #(value,...) instance_name
(port_connections);
```

You can also use a defparam statement to pass parameter values:

```
defparam instance_name.parameter_name = value;
```

Module Instance Example

```
module reg_4_bit (q, d, clock); output [3:0] q;
input [3.0] d;
input clock;
dff u1 (q[0],, d[0], clock;
dff u2(.clk(clock),.q(q[1]),.data(d[1]));
dff u3 (q[2],,d[2],clock);
defparam u3.delay = 3.2;
dff #(2) u4 (q[3],,d[3], clock);
endmodule
```

port order connection

port name connection

overriding parameter with defparam

overriding parameter in instantiation statement

```
module dff (q, qb, data, clk); output q, qb;
input data, clk;
parameter delay =1;
dff_udp #delay (q, data, clk);
not (qb, q);
endmodule
```

default delay stored as parameter

Procedural Blocks

```
type_of_block timing_control
statement_block : block_name
  local_register_declarations
  timing_control procedural_statements
end_of_statement_block
```

Types of Procedural Blocks

`initial`

A procedural block that executes once.

`always`

A procedural block that executes over and over again.

Statement Blocks

`begin - end`

Groups two or more statements that execute sequentially, so that all statements are evaluated in the order they are listed. Each timing control is relative to the execution of the previous statement.

`fork - join`

Groups two or more statements that execute in parallel, so that all statements are evaluated concurrently. Each timing control is absolute to when the group started.

`block_name`

(optional) When a statement block is given a name, it may have local registers and can be disabled.

`local_register_declarations`

(optional) Must be a register data type, and may only be declared if the statement block is given a name.

Timing Controls (Procedural Delays)

`#delay statement;`

Delays execution for a specific amount of time. The delay may be a literal number, a variable, or an expression.

`@(`*`edge net_or_register`*`) `*`statement;`*

Delays execution until there is a logic transition on a signal.

- *`edge`* (optional) must be either `posedge` or `negedge`. If no edge is specified, then any logic transition is used.
- The `or` keyword is used to specify events on any of more than one signal.

Examples	

`always@(posedge clk)`
`begin`
...
	always block executes when there is a rising edge on clk

`always@(a or b or c)`
`begin`
...
	always block execute whenever there is a transition on signals a, b, or c

`wait (`*`expression`*`);`

Statement that delays execution until the expression evaluates as true.

Procedural Assignments

You can use the following procedural assignments only in procedural blocks.

register = expression;

Blocking procedural assignment. VCS evaluates the expression and immediately assigns the value of the expression to the register, possibly blocking other evaluations at the same time step until after the assignment is completed. In the sequence:

```
begin
  reg1=reg2;
  reg2=reg1;
end
```

The first blocking assignment will change `reg1` before the second blocking assignment can evaluate `reg1`.

register <= expression;

Nonblocking procedural assignment. VCS evaluates the expression and schedules the assignment of the value of the expression at the end of the time step, after VCS evaluates and schedules the assignments of the other nonblocking procedural assignments in the time step. In the sequence:

```
begin
  reg1<=reg2;
  reg2<=reg1;
end
```

Both assignments are evaluated before `reg1` or `reg2` change value and VCS assigns to `reg2` the old value of `reg1`.

timing_control register = expression;

Delayed blocking procedural assignment. The timing control delays the evaluation and the execution of the assignment statement.

register = timing_control expression;

Blocking procedural assignment statement with intra-assignment delay. VCS evaluates the expression and schedules the assignment in a non-deterministic order in the time step specified by the timing control.

register <= timing_control expression;

Nonblocking procedural assignment statement with intra-assignment delay. VCS evaluates the expression and schedules the assignment for the end of the time step specified by the timing control.

assign *register = expression;*

Procedural continuous assignment. Overrides any other procedural assignments to a register.

deassign *register;*

Deactivates a procedural continuous assignment, allowing other assignments to again change the register.

force *net_or_register = expression;*

Forces any data type to a value, overriding all other logic. These statements impede performance.

release *net_or_register;*

Removes the effect of a force or deactivates a procedural continuous assignment.

Programming Statements

if (*expression*) *statement_or_statement_block*

Executes the next statement or statement block if the expression evaluates as true.

if *(expression) statement_or_statement_block*
else *statement_or_statement_block*

Executes the first statement or statement block if the expression evaluates as true; Executes the second statement or statement block if the expression evaluates as false.

case (*net_or_register*)
** *case_match1*: *statement_or_statement_block***
** *case_match2*,**
** *case_match3*: *statement_or_statement_block***
** default: *statement_or_statement_block***
endcase

Compares the net or register to each case and executes the statement or statement block associated with the first matching case, or executes the default if none of the cases match (the default case is optional).

casez

Special version of the `case` statement which allows a Z logic value in the case matches to represent don't-care bits.

casex

Special version of the `case` statement which allows Z or X logic values in the case matches to represent don't-care bits.

forever *statement_or_statement_block*

An infinite loop that continuously executes the statement or statement block.

repeat (*number*) *statement_or_statement_block*

A loop that executes the statement or statement block the specified number of times. Number can be an integer, net, register, module parameter, or an expression. VCS evaluates the number only once.

while(*expression*) *statement_or_statement_block*

A loop that executes a statement or statement block as long as an expression evaluates as true or non-zero.

for(*initial_assignment;expression; step_assignment*) *statement_or_statement_block*

1. Executes the `initial_assignment` once when the loop starts.
2. Executes the statement or statement block as long as the `expression` evaluates as true or non-zero.
3. Executes the `step_assignment` at the end of each pass through the loop.

disable *group_name_or_task_name;*

Discontinues execution of a named group of statements or a task. Simulation of that group jumps to the end of the group without executing any scheduled events.

Procedural Block Examples	

`initial`
`begin`
`  clk = 0;`
`  #1000 forever #25 clk=~clk;`
`end`
	A 50 ns clock oscillator that starts after 1000 time units

`always @(posedge clk)`
`begin`
`  word [15:8] <= word [7:0];`
`  word [7:0] <= word [15:8];`
`end`
	A sequential block that triggers on a rising edge on clk.
	Nonblocking assignments avoid race conditions in the byte swap.

`always @(a or b or mode)`
`if (mode == 0) y = a + b;`
`else y = a * b;`
	A combinational block that executes whenever any signal in the “sensitivity list” changes.

Procedural Block Examples	

`always @(posedge clk)`
`begin`
`casez (opcode)`
`4'b1???: alu_out = accum;`
`4'b0000: while (block_xfer)`
`repeat (5) @(posedge clk)`
`begin`
`RAM[address]=data_bus;`
`address=address + 1'`
`end`
`4'b0111:`
`begin:load`
`integer i;`
`for (i=0; i<=255;i=i+1)`
`@(negedge clk)`
`data_bus = RAM [i];`
`end`
`endcase`
`end`
	A sequential logic block that triggers off clk.
	casez makes Z don't care.
	The ? is the same as Z.
	Loop until block_xfer is false.
	Loop five clock cycles
	
	Named block.
	Local register

`always @(posedge clk)`
`fork`
`#5 count = count + 1;`
`#2.4 if (count == 4'd15)`
`look_ahead = 1;`
`else`
`look_ahead = 0`
`join`
	The fork causes parallel evaluation.
	count has 5 ns delay.
	Look ahead carry has 2.4 ns delay

Operators

Operators perform an operation on one or two operands:

Unary expression: *`operator operand`*

Binary expression: *`operand operator operand`*
The operands can be either net or register data types and can be scalar, vector, bit selects, or part selects from a vector.

Operator	Usage	Description
Arithmetic Operators		
`+`	`m + n`	Add n to m
`-`	`m - n`	Subtract n from m
`-`	`-m`	Negate m (2's complement)
`*`	`m * n`	Multiply m by n
`/`	`m / n`	Divide m by n
`%`	`m % n`	Modulo (remainder) of m / n
Bitwise Operators		
`~`	`~m`	Invert each bit of m
`&`	`m & n`	AND each bit of m with each bit of n
`|`	`m | n`	OR each bit of m with each bit of n
`^`	`m ^ n`	Exclusive OR each bit of m with n

~^ ^~	m ~^ n m ^~ n	Exclusive NOR all bits in m (1-bit result)
Unary Reduction Operators		
&	&m	AND all bits in m together (1-bit result)
~&	~&m	NAND all bits in m together (1-bit result)
|	|m	OR all bits in m together (1-bit result)
^	^m	Exclusive OR all bits in m (1-bit-result)
~^ ^~	~^m ^~m	Exclusive NOR all bits in m (1-bit result)
Logical Operators		
!	!m	Is m not true? (1-bit True/False result)
&&	m && n	Are both m and n true? (1-bit T/F result)
||	m || n	Are either m or n true? (1-bit T/F result)
Equality Operators (compare logic values 0 and 1)		
==	m == n	Is m equal to n? (1-bit True/False result)
!=	m != n	Is m not equal to n? (1-bit T/F result)
Identity Operators (compare logic values 0, 1, X, and Z)		
===	m === n	Is m identical to n? (1-bit True/False result.)
!==	m !== n	Is m not identical to n? (1-bit T/F result)
Relational Operators		
<	m < n	Is m less than n? (1-bit True/False result)
>	m > n	Is m greater than n? (1-bit T/F result)
<=	m <= n	Is m less than or equal to n? (1-bit result)
>=	m >= n	Is m greater than or equal to n? (1-bit result.)
Logical Shift Operators		
<<	m << n	Shift m left n-times
>>	m >> n	Shift m right n-times
Miscellaneous Operators		
? :	sel ? m:n	If sel is true, select m: else select n
{ }	{m,n}	Concatenate m to n, creating larger vector
{{}}	{n{m}}	Replicate m n-times
->	-> m	Trigger an event on an event data type

Operator Precedence	
! ~ + - & ~& | ^| ^ ~^	unary
* / %	binary
+ -	
<< >>	
< <= > >=	
== != === !==	
&	
^ ~^	
|	
&&	
||	
?:	ternary

Note: All operators with the same precedence associate left to right except the ternary operator.

Continuous Assignments

Continuous assignments model combinational logic.

- Continuous assignments are declared outside of procedural blocks. They automatically become active at time zero, and are evaluated concurrently with procedural blocks, module instances, and primitive instances.
- The signal on the left-hand side must be a net data type.
- Every time a signal changes on the right-hand side, the entire right-hand side is re-evaluated, and the result is assigned to the net on the left-hand side.

Explicit Continuous Assignments

***net_type* [*size*] *net_name*;**
assign #*delay net_name* = *expression*;

Requires two statements - one to declare the net, and one to continuously assign a value to it.

Implicit Continuous Assignments

***net_type* (*strength*) [*size*] #*delay net_name* = *expression*;**

Combines the net declaration and continuous assignment into one statement.

- *net_type* can be any of the net data types except `trireg`.

- *strength* (optional) may only be specified when the continuous assignment is combined with a net declaration. The default strength is `(strong0,strong1)`.
- *delay* (optional) follows the same syntax as primitive delays (See “Primitive Delays” on page 13.). The default is zero delay.
- *expression* may include any data type, any operator, and calls to functions.

Continuous Assignment Examples
`wire [31:0] mux_out; assign mux_out = sel ? a : b;` A 32-bit wide 2:1 MUX
`tri [0:15] #2.8 buf_out = en ? in: 16’bz` A 16-bit wide tri-state buffer with propagation delay
`wire [63:0] (strong1,pull0) alu_out = alu_function(opcode,a,b);` A 64-bit ALU with ECL output strengths

Task and Function Definitions

```
task task_name;
  input, output, and inout argument declarations;
  local register parameter or event declarations;
  procedural statement or statement block;
endtask

function [range or register type] function_name;
  input argument declarations;
  local register declarations;
  procedural statement or statement block;
endfunction
```

Tasks are subroutines, and can have any number of input, output, or inout arguments, and can contain timing controls (`#`, `@`, or `wait`). The order in which you declare the input, output, and inout arguments in this definition determines the order in which you specify arguments in the task enabling statement that calls the task.

Task Example	

`task read_mem;`
`input [15:0] address;`
`output [31:0] data;`
`begin`
`  read_request = 1;`
`  wait (read_grant) addr_bus=address;`
`  wait (data_ready) data=data_bus`
`  #5 addr_bus = 16”bz`
`  read_reaquest = 0;`
`end`
`endtask`
	This is a task definition.
	Note the order of the argument declarations.

`always @(posedge clk)` `read_mem(PC,IR);`	This is an always block for a task enabling statement. VCS matches PC and IR to arguments address and data.

Functions must have at least one input argument. Functions can not contain timing controls (`#`, `@`, or `wait`). Functions return the value assigned to the name of the function.
The order that you declare the arguments determines the order of the arguments when you call this function.

Example of a Function	

`function [7:0]fmux;`
`input [7:0] a,b;`
`input select;`
`begin`
`fmux=select?a,b;`
`end`
	Function definition

`val=fmux(i1,i2,sel);`	This is a procedural assignment that assigns the return value of the function.

Specify Blocks

```
specify
  specify_block_parameters;
  pin_to_pin_path_delays
  timing_constraint_checks
endspecify
```

Specify Block Parameters

`specparam = value;`

Specparams are constants used to store delays, delay calculation factors, synthesis factors, etc.

Pin-to-Pin Path Delays

Full connection path delay

`(input_port *> output_port) = (delay);`

Every bit of input has a delay path to every bit of output.

Parallel connection path delay

`(input_port => output_port) = (delay);`

Each bit of input is connected to the same bit of output (bit0 to bit0, bit1 to bit1 ...)

Level sensitive conditional path delays

`if (condition)(input_port=>output_port)=(delay);`

`if (condition)(input_port*>output_port)=(delay);`

`condition` must use input ports and expressions that resolve to true/false.

Edge sensitive path delays

```
(edge input_port=>(output_port:source))=(delay);

(edge input_port*>(output_port:source))=(delay);
```

- *edge* (optional) may be either `posedge` or `negedge`. If not specified, then all input transitions are used.
- *source* is the input port or value the output will receive, which may be different than the edge sensitive trigger.

State Dependent path delays

```
if(condition_1)(input*>output)=(delay);

if(condition_2)(input*>output)=(different_delay);

if(condition_3)(input*>output)=(different_delay);
```

The *conditions* must use input ports and expressions that resolve to true/false.

Polarity

You can specify a positive or negative polarity to the module path.

Prepend a + (plus) to the `*>` or `=>` symbol to specify positive polarity. In edge sensitive path delays, append the + to the `:` between the *output_port* and the *source*.

Prepend a – (minus) to the `*>` or `=>` symbol to specify negative polarity.In edge sensitive path delays, prepend the – to the `:` between the *output_port* and the *source*.

Delays for Different Transitions

You can specify different delays for 1,2,3, or 6 transitions. Each transition can have a single delay or a min:typ:max range of delays.

Number of Delays	Transitions Represented
1	all output transitions
2	rise, fall output transitions
3	rise, fall, turn-off output transitions
6	rise, fall, 0 to Z, Z to 1, 1 to Z, Z to 0

Specify Block Examples	Notes
`(a => b)=1.8;`	One delay for all output transitions
`(a -*> b)=2:3:4;`	One min:typ:max delay range for all output transitions. Port b receives the inverted value of a.
`specparam t1=3:4:6,t2=2:3:4;` `(a *> y) = (t1,t2);`	Different delays for rise, fall transitions
`(a *> y1, y2)=(2,3,4,3,4,3);`	Different delays for each output transition

Specify Block Examples	Notes
`if (clock)` `(in*>out) = 2:3:5;`	Level sensitive conditional path delay. Timing path only exists if clock is low (not true).
`(posedge clock=>(q +: data))=2;`	Edge sensitive path delay. Timing path is positive edge of clock to q. Port q receives the non-inverted value of data.
`if (rst && pst)` `(posedge clk=>(qb -: d)=(2, 3);`	Combined level and edge sensitive path delay. Port qb receives the inverted value of d.

`if (opcode=2'b00)(a,b *> o)=25;`
`if (opcode = 2'b01)(a,b *>o)=15;`
`if (opcode = 2'b10)(a,b *> o)=10;`
`(opcode*> o)=15;`
	State Dependent path delays. An ALU with different delays for each operation.

User Defined Primitives (UDPs)

User Defined Primitives allow you to define new primitives. You make instance declarations of UDPs exactly the same as built-in primitives.

```
primitive primitive_name (output, input, input, ...);
  output terminal_declaration;
  input terminal_declarations;
  reg output_terminal;
  initial output_terminal = logic_value;
  table
    table_entry;
    table_entry;
  endtable
endprimitive
```

Terminal Declarations

- Only one output is allowed, and must be the first terminal.
- The maximum number of inputs is 10 inputs.
- All terminals must be scalar (1-bit).

Declaring Sequential Logic UDPs

- `reg` data type declaration (optional) is used to define a sequential UDP by creating internal storage. Only the output terminal may be declared as a `reg`.
- `initial` (optional) is used to define the initial (power-up) state for sequential UDPs. Only the logic values 0, 1, and X may be used. The default state is X (unknown) if no initial state is specified.

UDP Table Entries

Combinational logic table entry

```
input_logic_values : output_logic_value;
```

Sequential logic table entry

```
input_logic_values : state : output_logic_value;
```

The state field specifies the current state or logic value of the output. A Sequential logic table requires a `reg` declaration for the output.

When writing table entries:

- The input values in the table entries must be listed in the same order as the terminal list in the UDP header.
- The high impedance state may not be used in a UDP. A logic Z on an input will be interpreted as a logic X (unknown).
- Any combination of input values not specified in the table result in a logic X (unknown) value on the output.
- Only one signal may have an edge transition specified for each entry in the table.
- If edge transitions are specified for one input, the UDP becomes sensitive to transitions on all inputs, therefore all other inputs must have a table entry to cover transitions, or when the transition occurs the UDP will output an X.
- Level sensitive table entries have precedence over edge sensitive table entries.

UDP Table Symbols

Symbol	Definition
`0`	Logic 0
`1`	Logic 1
`x` or `X`	Unknown
`?`	Don't care if 0, 1, or X
`b` or `B`	Don't care if 0 or 1
`(vw)`	Parentheses enclosing different logic values expresses a transition from the *v* to the *w* value. Substitute 0, 1, x, ?, or b for *v* and *w*.
`r` or `R`	Rising transition: same as `(01)`
`f` or `F`	Falling transition: same as `(10)`
`p` or `P`	Positive transition: same as `(01)`, `(0X)` or `(X1)`
`n` or `N`	Negative transition: same as `(10)`, `(1X)` or `(X0)`
`*`	Any possible transition: same as `(??)`
`-`	No change on output of a sequential device

UDP Examples

A combinational UDP. note that the order of the terminals in the UDP header match the order of the declarations for these terminals.
 	
`primitive mux (y, a, b, sel);`
`output y;`
`input a, b, sel;`
`table`
`//Table order for inputs matches primitive statement`
`//a b sel y`
`  0 ?  0  :0; //select a; don't care about b`
`  1 ?  0  :1; //select a; don't care about b`
`  ? 0  1  :0; //select b; don't care about a`
`  ? 1  1  :1; //select b; don't care about a`
`endtable`
`endprimitive`

A sequential UDP. Output terminal also declared as a `reg` which makes it an internal storage state.
 	
`primitive d_flip_flop (q, d, clk, rst);`
`output q;`
`input clk, rst, d;`
`reg q;`
`initial q = 0;`
`table`
`// d clk rest : state : q`
`  ?  ?  0  :  ?  :  0 ; //lo true reset`
`  0  r  1  :  ?  :  0 ; //clock in a 0`
`  1  r  1  :  ?  :  1 ; //clock un a 1`
`  ?  n  1  :  ?  :  - ; //ignore`
`  //negedge of clk`
`  *  ?  1  :  ?  :  - ; //ignore all`
`  //edges on d`
`  ?  ?  p  :  ?  :  - ; //ignore posedge`
`  //on rst`
`endtable`
`endprimitive`

System Tasks and Functions

`$assert_monitor`

Analogous to the standard `$monitor` system task in that it continually monitors specified assertions and displays what is happening with them (you can have it only display on the next clock of the assertion). Its syntax is as follows:

`$assert_monitor([0|1,]`*`assertion_identifier...`*`);`

Where:

`0`

Specifies reporting on the assertion if it is active (VCS is checking for its properties) and for the rest of the simulation reporting on the assertion or assertions, whenever they start.

`1`

Specifies reporting on the assertion or assertions only once, the next time they start.

If you specify neither 0 or 1, the default is 0.

assertion_identifier...

A comma separated list of assertions. If one of these assertions is not declared in the module definition containing this system task, specify it by its hierarchical name.

$assert_monitor_off

Disables the displat from the `$assert_monitor` system task.

$assert_monitor_on

Re-enables the display from the `$assert_monitor` system task.

$bitstoreal(*bit_value*)

System function that converts a bit pattern to a real number.

$countdrivers(*net* [,*net_is_forced*, *number_of_0_1_or_x_drivers*, *number_of_0_drivers*, *number_of_1_drivers*, *number_of_x_drivers*]);

System function that counts the number of drivers on a net.

$deposit(*net_or_variable*,*value*);

Deposits a value that overrides the value from any other driver of the net or variable. This value can be overridden by any subsequent simulation event. You can't deposit a value it a bit-select or part-select.

$disable_warnings[(*module_instance*,...)];

Disables the display of timing check warning messages but does not disable the toggling of notifier registers. Specifying an instance disables these messages for that instance and all instances hierarchically under it.

$display[b|h|o]("*text & format specifications*", *net_register_or_expression*,...);

Displays the arguments you specify in the order that you specify them. The default format specification is decimal, appending `b` changes this default to binary, `h` to hexadecimal, `o` to octal.

$dist_exponential(*seed, mean*);

System function that returns random numbers where the distribution function is exponential. The *`mean`* argument must be greater than 0.

Mean is the average of all the n points xi => mean =Sum(xi)/.

Variance = mean*mean.

$dist_normal(*seed, mean, standard_deviation*)

System function that returns random numbers were there is a specified mean value and a standard deviation for the random numbers. The seed is saved for the next invocation.
Mean is the average or all the n points xi => mean = Sum(xi)/n.
Variance = sd*sd => Sum ((xi - mean)*(xi-mean))/n.
Where mean and sd are the specified *mean* and *standard_deviation* arguments.

$dist_poison(*seed, mean*);

System function that returns random numbers where there is a specified *mean* value that must be greater than 0. Same as `$dist_normal` but the probability distribution function is different.
Mean is the average of all the n points xi => mean = Sum(xi)/n.
Variance = mean.

$dist_uniform(*seed, start, end*)

System function that returns random numbers uniformly distributed between the start and end arguments.

$dumpall;

Creates a checkpoint in the VCD file.

$dumpchange("*filename*");

Specifies to stop recording transition times and values in the current VCD file and to start recording in a new file.

$dumpfile ("*filename*");

Specifies the name of the VCD file you want VCS to record.

$dumpflush;

Emptied the VCD file buffer and writes all this data to the VCD file.

$dumpoff;

Stops recording value change information in the VCD file.

$dumpon;

Starts recording value change information in the VCD file.

$dumpports(*module_instance*,[*module_instance*,]" *filename*");

For creating an extended VCD file (VCDE file) as specified in the proposed IEEE Std. 1364-2000.

$dumpportsall("*filename*");

By default VCS writes to files specified in the `$lsi_dumpports` or `$dumpports` system task only when a signal changes value. The `$dumportsall` system task records the values of the ports in the module instances specified in the `$lsi_dumpports` or `$dumpports` system task whether there is a value change on these ports or not.

$dumpportsflush("*filename*");

When writing files specified in the `$lsi_dumpports` or `$dumpports` system task, VCS stores simulation data in a buffer during simulation from which it writes data to the file. If you want VCS to write all simulation data from the buffer to the file or files at a particular time, execute this `$dumpportsflush` system task.

$dumpportslimit(*filesize*,"*filename*");

Specifies the maximum file size of the file specified by the `$lsi_dumpports` or `$dumpports` system task.

$dumpportsoff("*filename*");

Suspends writing to files specified in `$lsi_dumpports` or `$dumpports` system tasks.

$dumpportson("*filename*");

Resume writing to the file after writing was suspended by a `$dumpportsoff` system task.

$dumpvars(*level_number*,*module_instance* | *net_or_reg*);

Specifies the signals whose transition times and values you want VCS to record in the VCD file.

$enable_warnings[(*module_instance*,...)];

Enables the display of timing check warning messages after the execution of the `$disable_warnings` system task. Specifying an instance enables these messages for that instance and all instances hierarchically under it.

$fclose(*multichannel_descriptor*);

Closes the channels to the open files that are specified by the multichannel descriptor.

$fdisplay[b|h|o](*multichannel_descriptor*, "*text & format specifications*", *net_register_or_expression*,...);

Works like the `$display` system task except that it writes to the file specified by the multichannel descriptor instead of to the standard output.

$fflush("*filename*");

Specifies immediate writing of VCD data from the operating system's dump file buffer to the VCD file.

$fflushall;

Writes to all VCD files.

$finish[(0|1|2)];

Ends simulation. The optional argument specify how much diagnostic information VCS displays when it executed this system task.
0 prints nothing.
1 prints simulation time and location.
2 also prints statistics about memory usage and CPU time.

**$fmonitor[b|h|o](*multichannel_descriptor,*
"*text & format specifications*",
net_register_or_expression,...);**

Works like the `$monitor` system task except that it writes to the file specified by the multichannel descriptor instead of to the standard output.

$fopen("*filename*");

System function that opens the specified file and returns a 32 bit unsigned multichannel descriptor that is uniquely associated with the file.

**$fstrobe[b|h|o](*multichannel_descriptor,*
"*text & format specifications*",
net_register_or_expression,...);**

Works like the `$strobe` system task except that it writes to the file specified by the multichannel descriptor instead of to the standard output.

**$fwrite[b|h|o](*multichannel_descriptor,*
"*text & format specifications*",
net_register_or_expression,...);**

Works like the `$write` system task except that it writes to the file specified by the multichannel descriptor instead of to the standard output.

$getpattern(*memory_element*)

System function that returns the value of a memory element.

$gr_waves (["*label*",]*net_or_reg,...*);

Produces a VCD file with the name grw.dump. You can specify a display label for a net or register.

$hold(*reference_event, data_event, limit [, notifier]*);

Reports a timing violation when the data event happens too soon after the reference event. The *reference_event* and *data_event* arguments are input ports. You can use the `negedge` or `posedge` keywords in the data and reference events. The *notifier* argument is a `reg` used as a flag. This system task can only be entered in specify blocks.

$itor(*int_value*)

System function that converts integers to real numbers.

$log[("*filename*")];

If a filename argument is included, this system task stops writing to the vcs.log file or the log file specified with the `-l` runtime option and starts writing to the file you specify. If the file name argument is omitted, this system task tells VCS to resume writing to the log file after writing to the file was suspended by the `$nolog` system task.

$lsi_dumpports(*module_instance*,"*filename*");

For LSI certification of your design, specifies recording the transition times and values of the ports in an instance and the name of the file where VCS writes this information.

$monitor[b|h|o]("*text & format specifications*", *net_register_or_expression*,...);

Repeatedly displays the arguments you specify in the order that you specify them when any net or register in your arguments change value. The default format specification is decimal, appending `b` changes this default to binary, `h` to hexadecimal, `o` to octal.

$monitoroff;

Disables the `$monitor` system task.

$monitoron;

Re-enables the `$monitor` system task after it was disabled with the `$monitoroff` system task.

$nolog;

Disables writing to the vcs.log file or the log file specified by the `-l` runtime option or `$log` system task.

**$period(*reference_event, limit*
[, *notifier*]);**

Reports a timing violation when the same edge occurs too soon on the reference signal. The *reference_event* argument must include either the posedge or negedge keyword along with an input port. The *notifier* argument is a reg used as a flag. This system task can only be entered in specify blocks.

$q_add(*q_id, job_id, inform_id, status*);

Places an entry on a queue for stochastic analysis.

**$q_exam(*q_id, q_stat_code, q_stat_value,*
status)**

Provides statistical information about activity at the queue.

$q_full(*q_id, status*);

Checks to see if there is room for another entry on a queue.

**$q_initialize(*q_id, q_type, max_length,*
status);**

Creates a new queue.

$q_remove(*q_id, job_id, inform_id, status*);

Receives an entry from a queue.

$random[(*seed*)]

System function that generates a random number.

**$readmemb ("*filename*", *memory_name*
[, *start_addr* [, *finish_addr*]]);**

Reads and loads binary data from the specified file to the specified memory.

**$readmemh ("*filename*", *memory_name*
[, *start_addr* [, *finish_addr*]]);**

Reads and loads hexadecimal data from the specified file to the specified memory.

$realtime

System function that returns a real number time.

$realtobits (*real_value*)

System function that passes bit patterns across module ports, converting a real number to a 64 bit representation.

$recovery(*edge_triggered_reference_event, data_event, limit* [, *notifier*]);

Reports a timing violation when a reference event happens too soon after a data event. This system task can only be entered in specify blocks.

$recrem(*reference_event, data_event, recovery_limit, removal_limit, notifier, timestamp_cond, timecheck_cond, delay_reference, delay_delta*);

Reports a timing violation if a data event occurs less than a specified time limit before or after a reference event. This system task can only be entered in specify blocks.

$reset[(*stop_value* [, *reset_value* [, *diagnostic_value*]])];

Resets the simulation time to 0.

A *stop_value* argument of 0 halts simulation after the reset, a non-zero argument does not halt simulation after the reset.

The *reset_value* is an integer that is returned by the $reset_value system function.

A non-zero *diagnostic_value* argument specifies diagnostic messages before the reset.

$reset_count

System function that keeps track of the number of times you have reset the simulation time.

$reset_value

System function that you can use to pass a value from before to after VCS executes the $reset system task.

$restart("*checkfile_name*");

Restarts the simulation from the point is was saved in the check file with the $save system task. Enter this system task at the CLI prompt.

$rtoi (*real_value*)

System function that converts real numbers to integers by truncating the real value.

$save ("*checkfile_name*");

Records the current simulation state in the specified check file. Enter, on a command line, *filename* instead of simv to restart simulation at the place you saved. You can also specify it at the CLI prompt with the $restart system task.

$sdf_annotate ("*sdf_file*"[, *module_instance*]
[,"*sdf_configfile*"][,"*sdf_logfile*"]
[,"*mtm_spec*"] [,"*scale_factors*"]
[,"*scale_type*"]);

Tells VCS to back annotate delay values from and SDF file to your Verilog design.

sdf_file	Path to the SDF file
module_instance	Instance where backannotation starts
sdf_configfile	Path to SDF configuration file
sdf_logfile	Path for backannotation log file
mtm_spec	Specify MINIMUM, TYPICAL,MAXIMUM, or TOOL_CONTROL
scale_factor	Colon separate string of three numbers. The are the multipliers for min:typ:max delay triplets.
scale_type	Specifies delays to use from triplets. Specify FROM_MINIMUM, FROM_TYPICAL, FROM_MAXIMUM, or FROM_MTM.

$setup (*data_event*, *reference_event*, *limit*
[, *notifier*]);

Reports a timing violation when the data event happens before and too close to the reference event. The *reference_event* and *data_event* arguments are input ports. you can also use the negedge or posedge keywords in the data or reference event. The *notifier* argument is a reg used as a flag. This system task can only be entered in specify blocks.

$setuphold (*reference_event*, *data_event*,
setup_limit*, *hold_limit* [, *notifier*,
***timestamp_cond*, *timecheck_cond*,**
***delayed_reference_signal*,**
***delayed_data_signal*]);**

Combines the $setup and $hold system tasks. You can use negative setup and hold limits. You can specify conditions that must be true for a timing violation and name delayed copy signals. The *timestamp_cond* argument is a condition that must be also true for an event on the reference signal to cause a violation. The *timecheck_cond* argument is a condition that must be also true for an event on the data signal to cause a violation. The *delayed_reference_signal* and *delayed_data_signal* arguments are copies of the reference and data signal with a propagation delay calculated from the setup and hold limits if one of these limits is negative.
This system task can only be entered in specify blocks.

**$skew (*reference_event, data_event, limit*
[, notifier]);**

Reports a timing violation if the interval between the reference event and the data event is more than the specified limit.
This system task can only be entered in specify blocks.

**$sreadmemb ("*filename*", *memory_name*,
start_addr, finish_addr,
string,[,*string*]);**

Works like the `$readmemb` system task except that it takes memory data values and addresses as string arguments.

**$sreadmemh ("*filename*", *memory_name*,
start_addr, finish_addr,
string,[,*string*]);**

Works like the `$readmemh` system task except that it takes memory data values and addresses as string arguments.

$stime

System function that returns an unsigned integer that is a 32 bit time.

$stop[(0|1|2)];

Halts simulation. The optional argument specify how much diagnostic information VCS displays when it executed this system task.
0 prints nothing.
1 prints simulation time and location.
2 also prints statistics about memory usage and CPU time.

**$strobe[b|h|o] ("*text & format specifications*",
net_register_or_expression,...);**

Displays the arguments you specify in the order that you specify them as the last simulation event in a time step. The default format specification is decimal, appending `b` changes this default to binary, `h` to hexadecimal, `o` to octal.

$system ("*command*");

Executes operating system commands.

$systemf("*command* %s ...","*string*",...);

System function that also executes operating system commands. It accepts multiple arguments including formatted strings.

$test$plusargs ("*plusarg_without_the_+*");

Checks for the existence of a given plusarg on the `simv` command line.

$time

System function that returns an integer that is a 64 bit time.

$timeformat (*units_number, precision_number, suffix_string, minimum_field_width*);

Specifies how the `%t` format specification reports time information.

The *`units_number`* argument is a range or integers between 0 to -15. 0 represents 1 second, -15 represents 1 fs. The intervening integers delineate orders of magnitude, for example -14 represents 10 fs, -13 represents 100 fs.

The *`precision_number`* argument specifies the number of decimal placed used in reporting time values.

The *`suffix_string`* argument is a text string that follows time values, for example, `ns`.

The *`mimimum_field_width`* arguments specifies how many characters, including spaces, are used to report time values.

$vcdplusautoflushoff;

Turns off the automatic “flushing” of simulation results to the VCD+ file whenever there is an interrupt such as when VCS executes the `$stop` system task.

$vcdplusautoflushon;

Tells VCS to “flush” or write all the simulation results in memory to the VCD+ file when ever there is an interrupt such as when VCS executes a `$stop` system task or when you halt the VCS using the “.” (period) CLI command or the Stop button in DVE.

$vcdplusclose;

Marks the current VCD+ file as completed, and close that file.

$vcdplusdeltacycleon;

Enables delta cycle recording in the VCD+ file for post-processing.

$vcdplusevent(*net_or_reg*,"*event_name*", "<E|W|I><S|T|D>");

Displays, in DVE, a symbol on the signal's waveform and in the Logic Browser. The *event_name* argument appears in the status bar when you click on the symbol.

E|W|I specifies severity. E for error, displays a red symbol, W for warning, displays a yellow symbol, I for information, displays a green symbol.

S|T|D specifies the symbol shape. S for square, T for triangle, D for diamond.

Enter no space between the E|W|I and the S|T|D arguments. Do not include angle brackets < >.

There is a limit of 244 unique events.

$vcdplusfile("*filename*");

Specifies the next VCD+ that DVE opens during simulation, after it executes the `$vcdplusclose` system task and when it executes the next `$vcdpluson` system task.

$vcdplusglitchon;

Turns on checking for zero delay glitches and other cases of multiple transitions for a signal at the same simulation time.

$vcdplusflush;

Tells VCS to "flush" or write all the simulation results in memory to the VCD+ file at the time VCS executes this system task. Use `$vcdplusautoflushon` to enable automatic flushing of simulation results to the file when simulation stops.

$vcdplusmemorydump(memory [,address [,second_address]]);

Records (dumps) in the VCD+ file the values in a memory, memory address, or a range of memory addresses.

$vcdplusoff[(*module_instance* | *net_or_reg*)];

Stops recording, in the VCD+ file, the transition times and values for the nets and registers in the specified module instance or individual nets or registers.

$vcdpluson[(*level_number*,*module_instance* | *net_or_reg*)];

Starts recording, in the VCD+ file, the transition times and values for the nets and registers in the specified module instance or individual nets or registers.

$vcdplustraceoff(*module_instance*);

Stops recording, in the VCD+ file, the order of statement execution in the specified module instance.

$vcdplustraceon[(*module_instance*)];

Starts recording, in the VCD+ file, the order of statement execution in the specified module instance and the module instances hierarchically under it.

**$width (*reference_event, limit, threshold*
[, *notifier*]);**

Reports a timing violation when a pulse is narrower than the specified limit. The *reference_event* must be an edge-triggered event (use posedge or negedge). The *threshold* argument is ignored.

**$write[b|h|o] ("*text & format specifications*",
net_register_or_expression,...);**

Works the same way as the $display system task except that $write does not automatically add a new line character to the end of its output.

**$writememb ("*filename*", *memory*
[, *start_address*] [, *end_address*]);**

Writes binary data in a memory to a file.

**$writememh ("*filename*",*memory*
[,*start_address*] [,*end_address*]);**

Writes hexadecimal data in a memory to a file.

Escape Sequences and Format Specifications

The escape sequences for printing special characters are as follows:

\n	The new line character
\t	The tab character
\\	The backward slash (\) character
\"	The double quotation mark (“) character
\ddd	A character specified by one to three octal digits
%%	The percent (%) character

The format specifications are as follows:

%h or %H	Display in hexadecimal format
%d or %D	Display in decimal format
%o or %O	Display in octal format
%b or %B	Display in binary format
%c or %C	Display in ASCII character format
%v or %V	Display net signal strength
%m or %M	Display hierarchical name

`%s` or `%S`	Display as a string
`%t` or `%T`	Display in current time format
`%e` or `%E`	Display real in exponential format
`%f` of `%F`	Display real in decimal format
`%g` or `%G`	Display real in decimal or exponential format, whichever results in shorter printed output

Compiler Directives

`celldefine

Specifies that the modules defined under this compiler directive are cell modules for certain PLI routines. See `endcelldefine on page 41.

`default_nettype wire | tri | tri0 | wand | triand | tri1 |wor | trior | trireg | none

Sets the default net data type for implicit nets.

`define *macro_identifier* [(*list_of_arguments*)] *macro_text*

Creates a macro for text substitution. Where the source code has `*macro_indentifier* entries, VCS substitutes the *macro_text*.

You can enter a *list_of_arguments*. These are formal arguments to be replaced by the actual arguments that accompany the `*macro_identifier* entries later in the source code.

`delay_mode_distributed

Ignore the module path delays specified in specify blocks in modules under this compiler directive, and use only the delay specifications on all gates, switches, and continuous assignments.

`delay_mode_path

For modules with specify blocks, ignore the delay specifications on all gates and switches declared under this compiler directive, and use only the module path delays and the delay specifications in continuous assignments.

`delay_mode_unit

Ignore the module path delays in specify blocks and change all the delay specifications on all gates, switches, and continuous assignments to the shortest time precision argument of all the `timescale compiler directives in the source code.

`delay_mode_zero

Change all the delay specifications on all gates, switches, and continuous assignments to zero and change all module path delays in specify blocks to zero.

`else

Use with the `ifdef compiler directive. Specifies that, if the macro specified in the `ifdef compiler directive is not defined, VCS should compile the source lines that follow as an alternative to those under the `ifdef compiler directive.

`endcelldefine

Specifies that the modules defined under this compiler directive are not cell modules. See `celldefine on page 40.

`endif

Use with the `ifdef and `else compiler directive. Specifies the end of the source code VCS compiles if a macro is or is not defined.

`endprotect

Specifies the end of the code you want encrypted. See `protect on page 41.

`endprotected

Specifies the end of the encrypted code in the source file output from source protection. See `protected on page 42.

`ifdef *macro_identifier*

Specifies compiling the source lines that follow if the specified text macro is defined by either the `define compiler directive or the +define compile-time option. See `undef on page 42, `else on page 41, and `endif on page 41.

`include *filename*

Inserts the contents of the specified source file into the source file that contains this compiler directive at the location of this compiler directive.

`noportcoerce

Disables the forcing of certain ports to inout ports. See `portcoerce on page 42.

`protect

Specifies the code you want encrypted.

`protected

Specifies the encrypted code in the source file output from source protection.

`portcoerce

Enables VCS to force certain ports declarations that follow to inout ports. VCS forces, for example, output ports whose values are assigned to signals in a module to an inout port because the assignment calls for the port to behave like an inout port. See `noportcoerce page 41.

`resetall

Resets all compiler directives to their default values.

`timescale *time_unit / time_precision*

Specifies the unit of measurement for time values and the accuracy to which VCS rounds time values.

`undef *macro_identifier*

Removes the text macro defined by the `define compiler directive. See `ifdef on page 41.

`uselib file = *filename* |
directory = *directory_name*

Specifies the library file or library directory to search for unresolved module instances. Enter path names if the library file or directory is not in the current directory. Include the keywords `file` or `directory`.

`vcs_mipdexpand

When back annotating SDF delay values from an ASCII text SDF file at runtime, this compiler directive enables the backannotation of delay values to individual bits of a port.
Similarly, a PLI application will not be able to pass delay values to individual bits of a port unless the port is under this compiler directive.

`vcs_mipdnoexpand

Turns off the enabling of backannotating delay values on individual bits of a port as specified by a previous `vcs_mipdexpand compiler directive.

Environment Variables

DISPLAY_VCS_HOME

Enables the display at compile time if the path to the directory specifies with the `VCS_HOME` environment.

LM_LICENSE_FILE

The complete path of the VCS license file or *port@host*.

PATH

On UNIX add `$VCS_HOME/bin` to this environment variable. variable.

TMPDIR

Specifies the directory for temporary compilation files.

VCS_HOME

Specifies the directory where you installed VCS.

VCSI_HOME

Specifies the directory where you installed VCSi.

VCS_NO_RT_STACK_TRACE

Tells VCS not to return a stack trace when there is a fatal error and instead dump a core file for debugging purposes.

VCS_CC

Specifies the C compiler.

VCS_COM

Specifies the path to the VCS compiler executable named vcs1 (or vcs1.exe).

VCS_LIC_EXPIRE_WARNING

By default VCS displays a warning 30 days before a license expires. You can specify that this warning begin fewer days before the license expires with this environment variable. To disable the warning, enter the 0 value:

VCS_LOG

Specifies the runtime log file name.

VCS_RUNTIME

Specifies which runtime library named libvcs.a VCS uses.

VCS_SWIFT_NOTES

Enables the printf PCL command.

VCS_WARNING_ATSTAR

Specifies the number of signals in a Verilog-2001 implicit sensitivity list that must be exceeded before VCS displays a warning. The default limit is 100 signals.

Compile-Time Options

+acc+1|2|3|4

Old style method to enable PLI ACC capabilities for the entire design.
1 enables all capabilites except value change callbacks, in other words breakpoints, and delay annotations.
2 enables what 1 enables plus breakpoints on signals.
3 enables what 2 enables plus annotating module path delays.
4 enables what 4 enables plus annotating gate delays.

+ad=*partition_filename*

Specifies the partition files used in mixed A/D simulation.

+allmtm

Allows you to specify at runtime which values in min:typ:max delay value triplets in compiled SDF files to use with `+maxdelay`, `+mindelays`, or `+typdelays` runtime options.

+applylearn+[*filename*]

Compiles your design to enable only the ACC capabilities that you needed for the debugging operations you did during a previous simulation of the design.
The `+vcs+learn+pli` runtime option records where you used ACC capabilities in a file named pli_learn.tab. If you do not change the file's name or location, you can omit `+filename` from this option

-as *assembler*

Specifies an alternative assembler. Not supported on IBM RS/6000 AIX.

-ASFLAGS *options*

Pass options to assembler. Not supported on IBM RS/6000 AIX.

-assert *keyword_argument*

The keyword arguments are as follows:

enable_diag

Enables further control of assertion results reporting with runtime options.

filter_past

Ignores assertion subsequences containing past operators that have not yet eclipsed the history threshold.

disable_cover

Disables coverage for cover statements.

dumpoff

Disables the dumping of SVA information in the VPD file during simulation.

+autoprotect[*file_suffix*]

Creates a protected source file; all modules are encrypted.

+auto2protect[*file_suffix*]

Create a protected source file that does not encrypt the port connection list in the module header; all modules are encrypted.

+auto3protect[*file_suffix*]

Creates a protected source file that does not encrypt the port connection list in the module header or any parameter declarations that precede the first port declaration; all modules are encrypted.

+bidir+1

Tells VCS to finish compilation when it finds a bidirectional registered mixed-signal net.

-C

Stops after generating the intermediate C or assembly code.

-c

Tells VCS to proceed with compiling the source files and generates the intermediate C, assembly, or object files, then compile or assemble the C or assembly code, but not to link. Use this option if you want to link by hand.

+charge_decay

Enables charge decay in `trireg` nets. Charge decay will not work if you connect the `trireg` to a transistor (bidirectional pass) switch such as `tran`, `rtran`, `tranif1`, or `rtranif0`.

-cc *compiler*

Specifies and alternative C compiler..

-CC *options*

Works the same as -CFLAGS.

-CFLAGS *options*

Pass options to C compiler. Multiple -CFLAGS are allowed. Allows passing of C compiler optimization levels.

+cli+[*module_name*=]1|2|3|4

Enable CLI debugging.
1 enables you to see the values of nets and registers and deposit values to registers.
2 also enables breakpoints on value changes of nets and registers.
3 also enables you to force a value on nets.
4 also enables you to force a value on a register.
You can specify a module to enable CLI debugging only for instances of the module.

+cliedit

Enables you to use the UNIX GNU command line editing interface for entering CLI commands.

-cm line|cond|fsm|tgl|path|assert

Specifies compiling for the specified type or types of coverage. The arguments specifies the types of coverage:

line

Compile for line or statement coverage.

cond

Compile for condition coverage.

fsm

Compile for FSM coverage.

tgl

Compile for toggle coverage.

path

Compile for path coverage.

branch

Compile for branch coverage

assert

Compile for SystemVerilog assertion coverage.

If you want VCS to compile for more than one type of coverage, use the plus (+) character as a delimiter between arguments, for example:

```
-cm line+cond+fsm+tgl
```

-cm_assert_hier *filename*

Limits assertion coverage to the module instances specified in *filename*. Specify the instances using the same format as VCS coverage metrics. If this option is not used, coverage is implemented on the whole design.

-cm_cond *arguments*

Modifies condition coverage as specified by the argument or arguments:

basic

Only logical conditions and no multiple conditions.

std

The default: only logical, multiple, sensitized conditions.

full

Logical and non-logical, multiple conditions, no sensitized conditions.

allops

Logical and non-logical conditions.

event

Signals in event controls in the sensitivity list position are conditions.

anywidth

Enables conditions that need more than 32 bits.

sop

Specifies condition SOP coverage. It also tells VCS that when it reads conditional expressions that contain the ^ bitwise XOR and ~^ bitwise XNOR operators, it reduces the expression to negation and logical AND or OR.

for

Enables conditions in for loops.

tf

Enables conditions in user defined tasks and functions.

You can specify more than one argument. If you do use the + plus delimiter between arguments, for example:

```
-cm_cond basic+allops
```

-cm_constfile *filename*

Specifies a file listing signals and 0 or 1 values. VCS compiles for line, line and condition coverage as if these signals were permanently at the specified values and you included the `-cm_noconst` option.

-cm_count

Enables cmView to do the following:

- In toggle coverage, not just whether a signal toggled from 0 to 1 and 1 to 0, but also the number of times it so toggled.
- In FSM coverage, not just whether an FSM reached a state, had such a transition, but also the number of times it did.
- In condition coverage, not just whether a condition was met or not, but also the number of times the condition was met.
- In Line Coverage, not just whether a line was executed, but how many times.

-cm_dir *directory_path_name*

Specifies and alternative name and location for the simv.cm directory.

-cm_fsmcfg *filename*

Specifies an FSM coverage configuration file.

-cm_fsmopt *keyword_argument*

The keyword arguments are as follows:

allowTemp

Allows FSM extraction when there is indirect assignment to the variable that holds the current state.

optimist

Specifies identifying illegal transitions when VCS extracts FSMs in FSM coverage. cmView then reports illegal transitions in report files.

report2StateFsms

By default VCS does not extract two state FSMs. This keyword tells VCS to extract them.

reportvalues

Specifies reporting the value transitions of the reg that holds the current state of a One Hot or Hot Bit FSM where there are parameters for the bit numbers of the signals that hold the current and next state.

reportWait

Enables VCS to monitor transitions when the signal holding the current state is assigned the same state value.

reportXassign

Enables the extraction of FSMs in which a state contains the X (unknown) value.

-cm_glitch *period*

When you use this compile-time option, during simulation, VCS ignores value changes during the specified glitch period. You specify the period with a non-negative integer.

-cm_hier *filename*

When compiling for line, condition, toggle or FSM coverage, specifies a configuration file that specifies the module definitions, instances and subhierarchies, and source files you want VCS either to exclude from coverage or exclusively compile for coverage.

-cm_ignorepragmas

Tells VCS to ignore pragmas for coverage metrics.

-cm_libs yv|celldefine

Specifies compiling for coverage source files in Verilog libraries when you include the `yv` argument. Specifies compiling for coverage module definitions that are under the `` `celldefine `` compiler directive when you include the `celldefine` argument. You can specify both arguments using the plus (+) delimiter.

-cm_line contassign

Specified enabling line coverage for continuous assignments.

-cm_name *filename*

As a compile-time or runtime option, specifies the name of the intermediate data files.

-cm_noconst

Tells VCS not to monitor for conditions that can never be met or lines that can never execute because a signal is permanently at a 1 0r 0 value.

-cm_pp [gui]|[batch]

Tells VCS to start cmView. It tells VCS to startcmView in batch mode to write reports by default.

gui

VCS starts the cmView graphical user interface to displaying coverage data.

batch

Specifies the default operation, writing reports in batch mode.

-cm_resetfilter *filename*

You can filter out of FSM coverage transitions in assignment statements controlled by `if` statements where the conditional expression (following the keyword `if`) is a signal you specify in the file. This filtering out can be for the specified signal in any module definition or in the module definition you specify in the file. You can also specify in the file the FSM and whether the signal is true or false.

-cm_tglfile *filename*

Specifies displaying at runtime a total toggle count for one or more subhierarchies specified by the top-level module instance entered in the file.

-cm_tgl mda

Enables toggle coverage for Verilog 2001 multidimensional arrays and SystemVerilog unpacked arrays. Not requires for packed SystemVerilog arrays.

-comp64

Compiles the design in 64 bit mode and creates a 32 bit executable for simulating in 32 bit mode.

-cpp

Specifies the C++ compiler.

+csdf+precompile

Precompiles your SDF file into a format that is for VCS to parse when it is compiling your Verilog code.

+csdf+precomp+dir+*directory*

Specifies the directory path where you want VCS to write the precompiled SDF file.

+csdf+precomp+ext+*ext*

Specifies an alternative to the "_c" character string addition to the filename extension of the precompiled SDF file.

-debug

Enbables the use of UCLI commands and DVE.

-debug_all

Enbables the use of UCLI commands and DVE. Also enables line stepping.

+define+*macro_name=value+*

Defines a text macro. Test for this definition in your Verilog source code using the `` `ifdef `` compiler directive.

+delay_mode_distributed

Ignore the module path delays and use only the delay specifications on all gates, switches, and continuous assignments.

+delay_mode_path

For modules with specify blocks, ignore the delay specifications on all gates and switches and use only the module path delays and the delay specifications on continuous assignments.

+delay_mode_unit

Ignore the module path delays and change all the delay specifications on all gates, switches, and continuous assignments to the shortest time precision argument of all the `` `timescale `` compiler directives in the source code.

+delay_mode_zero

Change all the delay specifications on all gates, switches, and continuous assignments to zero and change all module path delays to zero.

+deleteprotected

Allows overwriting of existing files when doing source protection.

-doc

Starts acroread to display the PDF file for the VCS/VCSi documentation navigator.

-e *new_name_for_main*

Specifies the name of your main() routine in your PLI application.

-f *filename*

Specifies a file that contains a list of pathnames to source files and compile-time options.

-F *filename*

Same as the `-f` option but allows you to specify a path to the file and the source files listed in the file do not have to be absolute pathnames.

-file *filename*

This option is for problems you might encounter with entries in files specified with the `-f` or `-F` options. This file can contain more compile-time options and different kinds of files. It can contain options for controlling compilation and PLI options and object files. You can also use escape characters and meta-characters in this file, like `$`, `` ` ``, and `!` and they will expand, for example:

```
-CFLAGS '-I$VCS_HOME/include'
/my/pli/code/$PROJECT/treewalker.o
-P /my/pli/code/$PROJECT/treewalker.tab
```

You can comment out entries in this file with the Verilog `//` and `/* */` comment characters.

`-full64`

Compiles the design in 64 bit mode and creates a 64 bit executable for simulating in 64 bit mode.

`-gen_asm`

Specifies generating intermediate assembly code. Not supported on IBM RS/6000 AIX.

`-gen_c`

Specifies generating intermediate C code. This is the default in IBM RS/6000 AIX.

`-gen_obj`

Generate object code; default on Linux and Solaris platforms.

`+incdir+`*`directory`*`+`

Specifies the directories that contain the files you specified with the `` `include `` compiler directive. You can specify more that one directory, separating each path name with the "+" character.

`-h`

Displays a succinct description of the most commonly used compile-time and runtime options.

`-ID`

Displays the hostid or dongle ID for your machine.

`-ignore` ***`keyword_argument`***

The keyword arguments are as follows:

`unique_checks`

Suppresses warning messages about SystemVerilog `unique if` and `unique case` statements.

`priority_checks`

Suppresses warning messages about SystemVerilog `priority if` and `priority case` statements.

all

Suppresses warning messages about SystemVerilog `unique if`, `unique case`, `priority if` and `priority case` statements.

-j*number_of_processes*

Specifies the number of processes to use for parallel compilation. There is no space between the "**j**" character and the number.

-l *filename*

(lower case L) Specifies a log file where VCS records compilation messages and runtime messages if you include the `-R`, `-RI`, or `-RIG` options.

-ld *linker*

Specifies an alternative linker. Only applicable in incremental compile mode, which is the default.

-LDFLAGS *options*

Pass options to the linker. Only applicable in incremental compile mode, which is the default.

+libext+*extension*

Specifies that VCS only search the source files in a Verilog library directory with the specified extension. You can specify more than one extension, separating each extension with the "+" character. For example, `+libext++.v` specifies searches library files with no extension and library files with the .v extension.
Enter this option when you enter the `-y` option.

+liborder

Specifies searching for module definitions in the libraries that follow, on the vcs command line, a library that contains an unresolved instance before searching the libraries that precede the library with the unresolved instance.

+librescan

Specifies always starting the search for unresolved module definitions with the first library specified on the vcs command line.

+libverbose

Tells VCS to display a message when it finds a module definition in a source file in a Verilog library directory that resolves a module instantiation statement that VCS read in your source files, a library file, or in another file in a library directory.

-line

Enables stepping through the code and source line breakpoints in DVE.

+lint=[no]ID|none|all,...

Enables or disables Lint messages about your Verilog code.

-lmc-swift

Enables the LMC SWIFT interface.

-lmc-swift-template *swift_model_name*

Generates a Verilog template for a SWIFT Model.

-l*name*

Links the *name* library to the resulting executable.

-load *shared_library*:*registration_routine*

Specifies the registration routine in a shared library for a VPI application

-Marchive=*number_of_module_definitions*

Tells the linker to create temporary object files that contain the specified number of module definitions. Use this option if there is a command line buffer overflow caused by too many object files on the linker command line.

+maxdelays

Use maximum value when min:typ:max values are encountered in delay specifications SDF files.

-Mdefine=*name=value*

Adds a definition to the makefile.

-Mdelete

Use this option for the rare occurrence when the `chmod -x simv` command in the make file can't change the permissions on an old simv executable. This option replaces this command with the `rm -f simv` command in the make file.

-Mdirectory=*directory*

Specifies the incremental compile directory. The default name is csrc and the default location is the current directory.

+memcbk

Enables callbacks for memories and multi-dimensional arrays (MDAs). Use this option if your design has memories or MDAs and you are doing any of the following:

- Writing a VCD or VPD file during simulation. For VCD files, at runtime, you must also enter the `+vcs+dumparrays` runtime option. For VPD files you must enter the `$vcdplusmemon` system task. VCD and VPD files are used for post-processing with DVE.
- Using the VCS/SystemC Interface
- Interactive debugging with DVE
- Writing an FSDB file for Debussy
- Using any debugging interface application - VCSD/PLI (acc/vpi) that needs to use value change callbacks on memories or MDAs. APIs like acc_add_callback, vcsd_add_callback, and vpi_register_cb need this option if these APIs are used on memories or MDAs.

+memopt[**+2**]

Applies optimizations to reduce memory. +2 spawns a second process for more optimizations.

-Mfilename=*prefix*

Base name (prefix) for C source and object files.

-Minclude=*filename*

Add an include statement to the Makefile.

+mindelays

Use minimum value when min:typ:max values are encountered in delay specifications and SDF files.

-Mldcmd=*value*

Format string used to invoke the linker directly.

-Mlib=*directory*

Specifies the directory where VCS looks for descriptor information to see if a module needs to be recompiled. Also specifies a central place for object files. You use this option for shared incremental compilation.

-Mloadlist=*value*

If *value* is `Yes`, directly invoke the linker to link programs.

-Mmakefile=*filename*

Name of generated makefile (default is Makefile).

-Mmakeprogram=*program*

Program used to make object (default is make).

-Mrelative_path

Use this option if your linker has a limitation on the length of the linker line in the make file. If you specify a relative path with the `-Mlib` option, the `-Mrelative_path` option does not expand the relative path to an absolute path on the linker line in the make file.

-Msrclist=*filename*

Name of source list file that lists all object files created by VCS.

+multisource_int_delays

Enables multisource interconnect delays.

-Mupdate[=0]

By default, VCS overwrites the Makefile between compilations. If you wish to preserve the Makefile between compilations, enter this option with the 0 argument.

Entering this option without the 0 argument, specifies the default condition, incremental compilation and updating the Makefile.

+nbaopt

Removes the intra-assignment delays from all the nonblocking assignment statements in your design.

+neg_tchk

Enables negative values in timing checks.

-negdelay

Enables the use of negative values in IOPATH and INTERCONNECT entries in SDF files.

+nocelldefinepli+0|1|2

For specifying what VCS records in the VCD+ file about nets and registers defined under the `` `celldefine `` compiler directive.

0 enables recording the transition times and values of nets and registers in all modules defined under the `` `celldefine `` compiler directive or defined in a library that you specify with the `-v` or `-y` compile-time options.

1 disables recording the transition times and values of nets and registers in all modules defined under the `` `celldefine `` compiler directive.

2 disables recording the transition times and values of nets and registers in all modules defined under the `` `celldefine `` compiler directive or defined in a library that you specify with the `-v` or `-y` compile-time options whether the modules in these libraries are defined under the `` `celldefine `` compiler directive or not.

+noerrorIOPCWM

Changes the error condition, when a signal is wider or narrower than the inout port to which it is connected, to a warning condition, thus allowing VCS to create the simv executable after displaying the warning message.

-noIncrComp

Disables incremental compilation.

+nolibcell

Do not define as a cell modules defined in libraries unless they are under the `` `celldefine `` compiler directive.

+nospecify

Suppresses module path delays and timing checks in specify blocks.

-notice

Enables verbose diagnostic messages.

+notimingcheck

Suppresses timing checks in specify blocks.

+nowarnTFMPC

Suppress the “Too few module port connections” warning messages during Verilog Compilation.

+no_notifier

Disables the toggling of the notifier register that you specify in some timing check system tasks.

+no_tchk_msg

Disables the display of timing check warning messages but does not disable the toggling of notifier registers in timing checks. This is also a runtime option.

-ntb

Enables the use of the OpenVera Testbench language constructs described in the *OpenVera Language Reference Manual: Native TestBench*.

-ntb_cmp

Compiles and generates the testbench shell (*file*.vshell) and shared object files. Use this option when compiling the .vr file separately from the design files.

-ntb_define *macro*

Specifies any OpenVera macro name on the command line. You can specify multiple macro names using the + delimiter.

-ntb_filext *.ext*

Specifies an OpenVera file extension. You can specify multiple filename extensions using the + delimiter.

-ntb_incdir *directory_path*

Specifies the include directory path for OpenVera files. You can specify multiple include directories using the + delimiter.

-ntb_noshell

Tells VCS not to generate the shell file.Use this option when you recompile a testbench.

-ntb_opts *keyword_argument*

The keyword arguments are as follows:

check

Reports error, during compilation or simulation, when there is an out-of-bound or illegal array access.

dep_check

Enables dependency analysis and incremental compilation. Detects files with circular dependencies and issues an error message when VCS cannot determine which file to compile first.

no_file_by_file_pp

By default, VCS does file by file preprocessing on each input file, feeding the concatenated result to the parser. This argument disables this behavior.

print_deps

Enter this argument with the `dep_check` argument. This argument tells VCS to display the dependencies for the source files on the screen or in the file that you specify.

tb_timescale=*value*

Specifies an overriding timescale for the testbench. The timescale is in the Verilog format (for example, `10ns/10ns`).

use_sigprop

Enables the signal property access functions. (for example, vera_get_ifc_name()).

vera_portname

Specifies the following:
The Vera shell module name is named vera_shell.
The interface ports are named *ifc_signal*.
Bind signals are named, for example, as: \if_signal[3:0].

You can enter more than one keyword argument, using the + delimiter, for example:

```
-ntb_opts use_sigprop+vera_portname
```

-ntb_shell_only

Generates only a .vshell file. Use this option when compiling a testbench separately from the design file.

-ntb_sfname *filename*

Specifies the filename of the testbench shell.

-ntb_sname *module_name*

Specifies the name and directory where VCS writes the testbench shell module.

-ntb_spath

Specifies the directory where VCS writes the testbench shell and shared object files. The default is the compilation directory.

-ntb_vipext *.ext*

Specifies an OpenVera encrypted-mode file extension to mark files for processing in OpenVera encrypted IP mode. Unlike the `-ntb_filext` option, the default encrypted-mode extensions .vrp, .vrhp are not overridden, and will always be in effect. You can pass multiple file extensions at the same time using the + delimiter.

-ntb_vl

Specifies the compilation of all Verilog files, including the design, the testbench shell file and the top-level Verilog module.

-o *name*

Specifies the name of the executable file that is the product of compilation. The default name is simv .

-O*number*

Specifies an optimization level for how VCS both writes and compiles intermediate files. The *number* can be 0-4. The 0 value applies to assembly, C, and object code files. The values 0-4 apply to C files.
On windows, use `-CFLAGS` instead.

+old_iopath

Tells VCS to replace negative module path delays in SDF files with the delay specified in a module's specify block instead of replacing with a 0 delay.

+old_ntc

Prevents other timing checks from using the delayed versions of the signals in the `$setuphold` and `$recrem` timing checks.

+oldsdf

Disable compiling the SDF file.

+optconfigfile+*filename*

Specifies the VCS Configuration file for using Radiant technology.

-ova_api

Enables access to assert and cover statements.

-ovac

Starts the OVA compiler for checking the syntax of OVA files that you specify on the vcs command line.

-ova_cov

Enables viewing results with functional coverage.

-ova_cov_events

Enables functional coverage reporting of expressions.

-ova_cov_hier *filename*

Limits functional coverage to the module instances listed in the specified file.

-ova_debug or -ova_debug_vpd

Enables viewing functional coverage results in DVE.

-ova_file *filename*

Specifies an OpenVera Assertions file. Not required if the filename has an `.ova` extension.

-ova_filter_past

For assertions that are defined with the past operator, ignore these assertions where the past history buffer is empty. For instance, at the very beginning of the simulation the past history buffer is empty. So, a check/forbid at the first sampling point and subsequent sampling points should be igorned until the past buffer has been filled with respect to the sampling point.

-ova_enable_diag

Enables runtime options for controlling functional coverage reports.

-ova_inline

Enables compiling OVA code that is written in Verilog source files.

-ova_lint

Enables general rules for the OVA linter.

-ova_lint_magellan

Enables Magellan rules for the OVA linter.

+overlap

Enables accurate simulation of multiple non-overlapping violation windows for the same signals specified with negative delay values in timing checks.

-override-cflags

Tells VCS not to pass its default options to the C compiler.

-override_timescale=*time_unit/time_precision*

Overrides the time unit and a precision unit for all the `` `timescale `` compiler directives in the source code and, like the `-timescale` option, provides a timescale for all module definitions that precede the first `` `timescale `` compiler directive. Do not include spaces when specifying the arguments to this option.

-P *pli.tab*

Specifies a PLI table file.

-parameters *filename*

Changes parameters specified in the file to values specified in the file. The syntax for a line in the file is as follows:

```
assign value path_to_parameter
```

The path to the parameter is similar to a hierarchical name except you use the forward slash character (/) instead of a period as the delimiters.

+pathpulse

Enables the search for the `PATHPULSE$` specparam in specify blocks.

-platform

Returns the name of the *platform* directory in your VCS installation directory.

+pli_unprotected

Enables PLI and CLI access to the modules in the protected source file being created (PLI and CLI access is normally disabled for protected modules).

+plusarg_save

Enter this option in the file that you specify with the `-f` option so that VCS passes to the simv executable the options beginning with a plus + character that follow in the file.

+plusarg_ignore

Also enter this option in the file that you specify with the `-f` option so that VCS does not pass to the simv executable the options that follow in the file. Use this option with the `+plusarg_save` option to specify that other options should not be passed.

-PP

Enables you to enter in your Verilog source code system tasks like `$vcdpluson` to create a VPD file during simulation. This option minimizes net data details for faster post-processing. Faster than a VPD file produced by the `-I` or `-RI` option.

+print+bidir+warn

Tells VCS to display a list of bidirectional registered mixed signal nets.

+prof

Specifies that VCS writes the vcs.prof file during simulation that tells you the module definitions, module instances, and Verilog constructs in your design that use the most CPU time.

+protect[*file_suffix*]

Creates a protected source file, only encrypting `` `protect/`endprotect `` regions.

+pulse_e/*number*

Flag as error and drive X for any path pulse whose width is less than or equal to the percentage of the module path delay specified by the *`number`* argument.

+pulse_int_e/*number*

Same as the `+pulse_e` option but only applies to interconnect delays.

+pulse_int_r/*number*

Same as the `+pulse_r` option but only applies to interconnect delays.

+pulse_on_event

Specifies that when VCS encounters a pulse shorter than the module path delay, VCS waits until the module path delay elapses and then drives an X value on the module output port and displays an error message.

+pulse_on_detect

Specifies that when VCS encounters a pulse shorter than the module path delay, VCS immediately drives an X value on the module output port, and displays an error message. It does not wait until the module path delay elapses.

+pulse_r/*number*

Reject any pulse whose width is less than *number* percent of module path delay.

+putprotect+*target_dir*

Specifies the target directory for protected files.

-pvalue+*parameter_hierarchical_name=value*

Changes the specified parameter to the specified value.

-q

Suppresses VCS compiler messages.

-R

Run the executable file immediately after VCS links together the executable file. You can add any runtime option to the `vcs` command line.

+race

Specifies that VCS generate a report, during simulation, of all the race conditions in the design and write this report in the race.out file.

+race=all

Analyzes the source code during compilation to look for coding styles that cause race conditions.

+racecd

Specifies that VCS generate a report, during simulation, of the race conditions in the design between the `` `race `` and `` `endrace `` compiler directives and write this report in the race.out file.

+race_maxvecsize=*size*

Specifies the largest vector signal for which the dynamic race detection tool looks for race conditions.

+rad

Performs Radiant technology optimizations on your design.

-s

Stop simulation just as it begins. Use this option with the `-R` and `+cli` options.

+sdf_nocheck_celltype

Tells VCs not to check to make sure that the CELLTYPE entry in the SDF file does not match the module identifier for a module instance before back annotating delay values from the SDF file to the module instance.

+sdfprotect[*file_suffix*]

Creates a protected SDF file.

-sv_pragma

Tells VCS to compile the SystemVerilog assertions code that follows the `sv_pragma` keyword in a single line or mult-line comment.

-sysc

Tells VCS to look in the ./csrc directory for the subdirectories containing the interface and wrapper files needed by the VCS/SystemC cosimulation interface to connect the Verilog and SystemC parts of the design.

+systemverilogext+*ext*

Specifies a filename extension for SystemVerilog source files. If you use a different filename extesion for the SystemVerilog part of your source code, and this option, you can omit the `-sverilog` option.

-sverilog

Enables the use of the Verilog language extensions in the Accellera SystemVerilog specification.

-syslib *libs*

Specifies system libraries to be linked with the runtime executable.

+tetramax

Enter this option when simulatingTetraMAX's testbench in zero delay mode.

-timescale=*time_unit/time_precision*

If only some source files contain the `` `timescale `` compiler directive and the ones that don't appear first on the vcs command line, use this option to specify the time scale for these source files.

+timopt+*clock_period*

Starts the Timopt timing optimizer. the *+clock_period* argument specifies the clock period of the fastest clock in the design. Timopt applies timing optimizations to your design. Timopt also writes a timopt.cfg file in the current directory. This file contains clock signals and module definitions of sequential devices it's not sure of. You edit this file and recompile without the *+clock_period* argument to obtain more Timopt optimizations.

+transport_int_delays

Enables transport delays with full pulse control for single source nets.

+transport_path_delays

Turns on the transport behavior for I/O paths.

+typdelays

Use typical value when min:typ:max values are encountered in delay specifications and SDF files.

-u

Changes all characters in identifiers to uppercase.

-ucli

Specifies UCLI mode at runtime.

-use_vpiobj

Used to specify the vpi_user.c file that enables you to use the vpi_register_systf VPI access routine.

-V

Enables the display of compilation warning messages.

-v *filename*

Specifies a Verilog library file to search for module definitions.

+v2k

Enables the use of new Verilog constricts in the 1364-2001 standard.

+vc[+abstract][+allhdrs][+list]

Enables the direct call of C/C++ functions in your Verilog code using the DirectC interface. The optional suffixes specify the following:

+abstract

Specifies that you are using abstract access through vc_handles to the data structures for the Verilog arguments.

+allhdrs

Writes the vc_hdrs.h file that contains external function declarations that you can use in your Verilog code.

+list

Displays on the screen all the functions that you called in your Verilog source code.

-vcd2vpd *vcd_filename vcdplus_filename*

Tells VCS to find and run the vcd2vpd utility that converts a VCD file to a VCD+ file. VCS inputs to the utility the specified VCD file and the utility outputs the specified VCD+ file.

+vcs+boundscheck

Changes reading from or writing to an undeclared bit to an error condition instead of a warning condition.

+vcs+dumpvars

A substitute for entering the `$dumpvars` system task, without arguments, in your Verilog code.

+vcs+flush+log

Increases the frequency of flushing both the compilation and simulation log file buffers.

+vcs+flush+dump

Increases the frequency of flushing all the buffers for VCD files.

+vcs+flush+fopen

Increases the frequency of flushing all the buffers for files opened by the `$fopen` system function

+vcs+flush+all

Shortcut option for entering all three of the `+vcs+flush+log`, `+vcs+flush+dump`, and `+vcs+flush+fopen` options.

+vcs+initmem+0|1|x|z

Initializes all bits of all memories in the design.

+vcs+initreg+0|1|x|z

Initializes all bits of all regs in the design.

+vcsi+lic+vcs

Checks out a VCS license to run VCSi when all VCSi licenses are in use

+vcs+lic+vcsi

Checks out three VCSi licenses to run VCS.

+vcs+lic+wait

Tells VCS to wait for a network license if none is available.

+vcsi+lic+wait

Tells VCSi to wait for network license if none is available when the job starts

+vcs+mipdexpand

When back annotating SDF delay values from an ASCII text SDF file at runtime, as specified by the +oldsdf compile-time option which disables compiling the SDF file during compilation, if the SDF file contains PORT entries for the individual bits of a port, using this compile-time option enables VCS to backannotate these PORT entry delay values.

Similarly, using this compile-time option enables a PLI application to pass delay values to individual bits of a port.

-vera

Specifies the standard VERA PLI table file and object library.

-vera_dbind

Specifies the VERA PLI table file and object library for dynamic binding.

+verilog1995ext+*ext*

Specifies a filename extension for Verilog 1995 source files.

+verilog2001ext+*ext*

Specifies a filename extension for Verilog 2001 source files. If you use a different filename extesion for the Verilog 2001 part of your source code, and this option, you can omit the `+v2k` option.

+vhdllib+*logical_libname*[+*logical_libname...*]

This option specifies the VHDL logical library to use for VHDL design entity instances that you instantiate in your Verilog design. You can specify more than one logical library, separating them with the plus + delimiter.

+vissymbols

Makes symbols visible when you use the prof or gprof program to profile generated code.

-vpd2vcd *vcdplus_filename vcd_filename*

Tells VCS to find and run the vpd2vcd utility that converts a VCD+ file to a VCD file. VCS inputs to the utility the specified VCD file and the utility outputs the specified VCD+ file.

+vpi

Enables the use of VPI PLI access routines.

-Vt

Enables warning messages and displays the time used by each command.

+warn=[no]ID|none|all,...

Enables or disables warning messages.

+warn2val

Enable warning messages about 2 state simulation.

-y *directory_pathname*

Specifies a Verilog library directory to search for module definitions.

Runtime Options

-a *filename*

Specifies appending all messages from simulation to the bottom of the text in the specified file as well as displaying these messages in the standard output.

-assert *keyword_argument*

The keyword arguments are as follows:

dumpoff

Disables the dumping of SVA information in the VPD file during simulation.

filter

Blocks reporting of trivial implication successes. These happen when an implication construct registers a success only because the precondition (antecedent) portion is false (and so the consequence portion is not checked). With this option, reporting only shows successes in which the whole expression matched.

finish_maxfail=*N*

Terminates the simulation if the number of failures for any assertion reaches *N*. *N* must be supplied, otherwise no limit is set.

global_finish_maxfail=*N*

Stops the simulation when the total number of failures, from all SystemVerilog assertions, reaches *N*.

maxcover=*N*

Disables the collection of coverage information for cover statements after the cover statements are covered *N* number of times. *N* must be a positive integer, it can't be 0.

maxfail=*N*

Limits the number of failures for each assertion to *N*. When the limit is reached, the assertion is disabled. *N* must be supplied, otherwise no limit is set.

maxsuccess=*N*

Limits the total number of reported successes to *N*. *N* must be supplied, otherwise no limit is set. The monitoring of assertions continues, even after the limit is reached.

nocovdb

Tells VCS not to write the *program_name*.db database file for assertion coverage.

nopostproc

Disables the display of the SVA coverage summary at the end of simulation.

quiet 0 | 1

0 Disables messages, in standard output, about assertion failures.

1 Disables messages, in standard output, about assertion failures, but displays a summary of them at the end of simulation. The never triggered assertions are also reported.

report[=*path/filename*]

Generates a report file in addition to printing results on your screen. By default this file's name and location is ./assert.report, but you can change it to where you want by entering the filename path name argument.
The filename can start with a number or letter. The following special characters are acceptable in the filename: `%`, `^`, and `@`. Using the following unacceptable special characters: `#`, `&`, `*`, `[]`, `$`, `()`, or `!` has the following consequences:

- A filename containing `#` or `&` results in a filename truncation to the character before the `#` or `&`.
- A filename containing `*` or `[]` results in a `No match` message.
- A filename containing `$` results in an `Undefined variable` message.
- A filename containing `()` results in a `Badly placed ()'s` message.
- A filename containing `!` results in an `Event not found` message.

`success`

Enables reporting of successful matches in addition to failures. The default is to report only failures.

`verbose`

Adds more information to the end of the report specified by the `report` keyword argument and a summary with the number of assertions present, attempted, and failed.

You can enter more than one keyword, using the plus `+` separator, for example:

`-assert maxfail=10+maxsucess=20+success+filter`

`+cliecho`

Specifies that VCS displays CLI commands in a file that you specify with the `-i` option as VCS executes these commands. UNIX only.

`-cm line|cond|fsm|tgl|path|assert`

Specifies monitoring for the specified type or types of coverage. The arguments specifies the types of coverage:

`line`

Monitor for line or statement coverage.

`cond`

Monitor for condition coverage.

`fsm`

Monitor for FSM coverage.

`tgl`

Monitor for toggle coverage.

path

Monitor for path coverage.

branch

Monitor for branch coverage.

assert

Monitor for SystemVerilog assertion coverage.

If you want VCS to monitor for more than one type of coverage, use the plus (+) character as a delimiter between arguments, for example:

```
-cm line+cond+fsm+tgl
```

-cm_assert_dir *path/filename*

Specifies the path and filename of an initial coverage file. An initial coverage file is needed to set up the database. By default, an empty coverage file is loaded from the following directory: simv.vdb/snps/fcov.

-cm_assert_name *path/filename*

Specifies the file name or the full path name of the assertion coverage report file. This option overrides the default report name and location, which is ./simv.vdb/fcov/results.db If only a file name is given, the default location is used resulting in: ./*simv*.vdb/fcov/*filename*.db.

-cm_dir *directory_path_name*

Specifies and alternative name and location for the simv.cm directory.

-cm_glitch *period*

Specifies a glitch period during which VCS does not monitor for coverage caused by value changes. The period is an interval of simulation time specified with a non-negative integer.

-cm_log *filename*

Specifies a log file for monitoring for coverage during simulation.

-cm_name *filename*

As a compile-time or runtime option, specifies the name of the intermediate data files. On the `cmView` command line, specifies the name of the report files.

-cm_tglfile *filename*

Specifies displaying at runtime a total toggle count for one or more subhierarchies specified by the top-level module instance entered in the file.

+cm_zip

Required when using coverage metrics along with the SWIFT interface to VMC models or SmartModels or when Undertow dumping is enabled.

-E *program*

Starts the program that displays the compile-time options that were on the vcs command line when you created the simv (or simv.exe or some other name specified with the `-o` option) executable file.

-gui

Starts the DVE gui.

-grw *filename*

Sets the name of the `$gr_waves` output file to the specified file. The default filename is grw.dump.

-i *filename*

Specifies a file containing CLI commands that VCS executes when simulation starts.

-k *filename* | off

Specifies an alternative name or location for the vcs.key file into which VCS writes the CLI commands that you enter during simulation. The `off` argument tells VCS not to write this file.

-l *filename*

Specifies writing all messages from simulation to the specified file as well as displaying these messages in the standard output. This option begins with the letter "`l`" (lowercase "`L`") for log file.

+maxdelays

Species using the compiled SDF file for maximum delays generated wit hthe `+allmtm` compile-time option.
Also specifies using maximum delays for SWIFT VMC or SmartModels or Synopsys hardware models if you also enter the `+override_model_delays` runtime option.

+mindelays

Specifies using the compiled SDF file for minimum delays generated wit hthe `+allmtm` compile-time option.
Also specifies using minimum delays for SWIFT VMC or SmartModels or Synopsys hardware models if you also enter the `+override_model_delays` runtime option.

+no_notifier

Suppresses the toggling of notifier registers that are optional arguments of timing check system tasks.

+no_pulse_msg

Suppresses pulse error messages, but not the generation of StX values at module path outputs when a pulse error condition occurs.

+no_tchk_msg

Disables the display of timing check warning messages but does not disable the toggling of notifier registers in timing checks. This is also a compile-time option.

+notimingcheck

Suppress timing checks.

+ntb_cache_dir=*path_name_to_directory*

Specifies the directory location of the cache that VCS maintains as an internal disk cache for randomization.

+ntb_debug_on_error

Causes the simulation to stop immediately when a simulation error is encountered. In addition to normal verification errors, This option halts the simulation in case of runtime errors as well.

+ntb_enable_solver_trace=*value*

Enables a debug mode that displays diagnostics when VCS executes a randomize() method call. Allowed values are:

0 - Do not display (default).

1- Displays the constraints VCS is solving.

2 - Displays the entire constraint set.

+ntb_enable_solver_trace_on_failure=*value*

Enables a mode that displays trace information only when the VCS constraint solver fails to compute a solution, usually due to inconsistent constraints. When the value of the option is 2, the analysis narrows down to the smallest set of inconsistent constraints, thus aiding the debugging process. Allowed values are 0, 1, 2. The default value is 2.

+ntb_exit_on_error[=*value*]

Causes VCS to exit when value is less than 0. The value can be:

0: continue

1: exit on first error (default value)

N: exit on nth error.

When value = 0, the simulation finishes regardless of the number of errors.

+ntb_load=*path_name_to_libtb.so*

Specifies loading the testbench shared object file libtb.so.

+ntb_random_seed=*value*

Sets the seed value used by the top level random number generator at the start of simulation. The random(seed) system function call overrides this setting. The value can be any integer number.

+ntb_solver_mode=*value*

Allows choosing between one of two constraint solver modes. When set to 1, the solver spends more pre-processing time in analyzing the constraints, during the first call to randomize() on each class. When set to 2, the solver does minimal pre-processing, and analyzes the constraint in each call to randomize(). Default value is 2.

+ntb_stop_on_error

Causes the simulation to stop immediately when a simulation error is encountered, turning it into a cli debugging environment. In addition to normal verification errors, ntb_stop_on_error halts the simulation in case of run time errors. The default setting is to execute the remaining code within the present simulation time.

-ova_cov

Enables functional coverage reporting.

-ova_cov_name *filename*

Specifies the file name or the full path name of the functional coverage report file. This option overrides the default report name and location. If only a file name is given, the default location is used resulting in ./simv.vdb/fcov/*filename*.db.

-ova_cov_db *filename*

Specifies the path name of the initial coverage file. The initial coverage file is needed to set up the database. By default, an empty coverage file is loaded from simv.vdb/snps/fcov/default.db.

-ova_filter

Blocks reporting of trivial if-then successes. These happen when an if-then construct registers a success only because the if portion is false (and so the then portion is not checked). With this option, reporting only shows successes in which the whole expression matched. This option is enabled by the `-ova_enable_diag` compile-time option.

-ova_max_fail *N*

Limits the number of reported failures for each assertion to ***N***. When the limit is reached, the assertion is disabled. This option is enabled by the `-ova_enable_diag` compile-time option.

-ova_max_success *N*

Limits the number of successes for each assertion to ***N***. The monitoring of assertions continues, even after the limit is reached. This option is enabled by the `-ova_enable_diag` compile-time option.

-ova_report [*filename*]

Specifies writing an OpenVera Assertions report file. The default file name and location is simv.vdb/report/ova.report but you can specify a different name and location as an argument to this option.

-ova_simend_max_fail *N*

Terminates the simulation if the number of failures for any assertion is reached. This option is enabled by the `-ova_enable_diag` compile-time option.

-ova_success

Enables the reporting of successful matches. This option is enabled by the `-ova_enable_diag` compile-time option.

-ova_quiet [1]

Disables displaying functional coverage results on the screen. The optional 1 argument specifies displaying a summary of these results.

-ova_verbose

Adds more information to the end of the report including assertions that never triggered and attempts that did not finish, and a summary with the number of assertions present, attempted, and failed.

+override_model_delays

Enables you to use the `+mindelays`, `+typdelays`, or `+maxdelays` runtime options to specify timing for SWIFT SmartModels or Synopsys hardware models.

-q

Quiet mode. Suppress printing of VCS header and summary information, the proprietary message at the beginning of simulation, and the VCS Simulation Report at the end of simulation (time, CPU time, data structure size, and date)

-s

Stops simulation just as it beings, and enters interactive mode. Use with the `+cli+`*`number`* option.

+sdfverbose

Enables the display of more than ten warning and ten error messages about SDF back annotation.

+typdelays

Specifies using the compiled SDF file for typical delays generated wit hthe `+allmtm` compile-time option.
Also specifies using typical delays for SWIFT VMC or SmartModels or Synopsys hardware models if you also enter the `+override_model_delays` runtime option.

-ucli

Enables the use of UCLI commands.

-V

Verbose mode. Print VCS version and extended summary information.
Prints VCS compile and run-time version numbers, and copyright information, at start of simulation.

-vcd *filename*

Sets the output VCD file name to the specified file.
The default filename is verilog.dump.
A `$dumpfile` system task in the Verilog source code will override this option.

+vcs+dumparrays

Enables recording memory and multi-dimensional array values in the VCD file. You must also have used the `+memcbk` compile-time option.

+vcs+dumpoff+*t*+*ht*

Turn off value change dumping (`$dumpvars` system task) at time *t*. *ht* is the high 32 bits of a time value greater than 32 bits.

+vcs+dumpon+*t*+*ht*

Suppress `$dumpvars` system task until time *t*. *ht* is the high 32 bits of a time value greater than 32 bits.

+vcs+finish+*t*+*ht*

Finish simulation at time *t*. *ht* is the high 32 bits of a time value greater than 32 bits (optional).

+vcs+grwavesoff

Suppress `$gr_waves` system tasks.

+vcs+ignorestop

Tells VCS to ignore the `$stop` system tasks in your source code.

+vcs+flush+log

Increases the frequency of flushing both the compilation and simulation log file buffers.

+vcs+flush+dump

Increases the frequency of flushing all the buffers for VCD files.

+vcs+flush+fopen

Increases the frequency of flushing all the buffers for files opened by the `$fopen` system function

+vcs+flush+all

Shortcut option for entering all three of the `+vcs+flush+log`, `+vcs+flush+dump`, and `+vcs+flush+fopen` options.

+vcs+learn+pli

Keeps track of where you use ACC capabilities for debugging operations so that you can recompile your design and in the next simulation enable them only where you need them.
With this option VCS writes the pli_learn.tab secondary PLI table file. You input this file when you recompile your design with the `+applylearn` compile-time option.

+vcs+lic+vcsi

Checks our three VCSi licenses to run VCS.

+vcsi+lic+vcs

Checks out a VCS license to run VCSi when all VCSi licenses are in use.

+vcs+lic+wait

Tells VCS to wait for network license if none is available when the job starts.

+vcsi+lic+wait

Tells VCSi to wait for network license if none is available when the job starts

+vcs+mipd+noalias

If during a simulation run, acc_handle_simulated_net is called before MIPD annotation happens, a warning message is issued. When this happens you can use this option to disable such aliasing for all ports whenever mip, mipb capabilities have been specified. This option works for regular sdf annotation and not for compiled SDF.

+vcs+nostdout

Disables all text output from VCS including messages and text from `$monitor` and `$display` and other system tasks. VCS still writes this output to the log file if you include the `-l` option.

+vcs+stop+*t*+*ht*

Stop simulation at time t. ht is the high 32 bits of a time value greater than 32 bits (optional).

+vera_load=*filename*.vro

Specifies the VERA object file.

+vera_mload=*filename*

Specifies a text file that contains a list of VERA object files.

+vpdbufsize+*MB*

VCS uses an internal buffer to store value changes before it writes them to the VCD+ file on disk. VCS makes this buffer size either 5 MB or large enough to record 15 value changes for all nets and registers in your design, which ever is larger.
You can use this option to override the buffer size that VCS calculates for the buffer size. You specify a buffer size in megabytes.

+vpddrivers

Tells VCS to record the values of all the drivers of all the nets.

+vpdfile+*filename*+start+*start_time*+end+*end_time*

In post-processing, specifies the VCD+ file you wish to view in DVE. The optional `+start+start_time` and `+end+end_time` arguments specify you only want DVE to display the results from between these simulation times.

+vpdfilesize+*MB*

Specifies the maximum size of the VCD+ file. When VCS reaches this limit, VCS overwrites the oldest simulation history data in the file with the newest

+vpdignore

Tells VCS to ignore `$vcdplus` system tasks so VCS does not write a VCD+ file.

+vpdnocompress

Disables the automatic compressing of the data in VCD+ files

+vpdnostrengths

Disables recording strength information in the VCD+ file.

+vpdports

Tells VCS to record, in the VCD+ file, the port direction of signals that are ports.

+vpdupdate

If VCS is writing a VCD+ file during simulation, this option enables you to have VCS halt writing to the VCD+ file while the simulation is running and so that you can view the recorded results in DVE.

PLI Table Format

The syntax for a line in a PLI table file is as follows:

***$name PLI_specifications* [*ACC_capabilities*]**

Where:

$name	Is the name of your user-defined system task or system function
PLI_specifications	Is one or more specifications such as the name of the C function VCS calls when it executes the user defined system task or system function.
ACC_capabilities	Specifications for ACC capabilities to be added, removed, or changed from various parts of the design hierarchy.

The PLI Specifications

The valid PLI specifications are as follows:

`call=`*`function`*
: Specifies the name of call function. This specification is required.

`check=`*`function`*
: Specifies the name of check function.

`misc=`*`function`*
: Specifies the name of misc function.

`data=`*`integer`*
: Specifies the data value passed as first argument to call, check, and misc routines. The default is 0. See The IEEE Std 1364-1995 Section 17.6.1 for more information.

`size=`*`number`*
: Specifies the size of returned value in bits. This specification is required for user-defined system functions.

`args=`*`number`*
: Specifies the number of arguments to the user-defined system task or system function.

`minargs=`*`number`*
: Specifies the minimum number or arguments.

`maxargs=`*`number`*
: Specifies the maximum number or arguments.

`nocelldefinepli`
: Disables the dumping of value change and simulation time data, from modules defined under the `` `celldefine `` compiler directive, into the VCD+ file created by the `$vcdpluson` system task.

`persistent`
: Enables you to enter the user-defined system task on the CLI command line without including any of the `+cli` compile time options.

Specifying ACC Capabilities

The format for specifying ACC capabilities is as follows:

`acc=|+=|-=|:=`*`capabilities`*`:`*`module_names`*`[+]|%CELL|%TASK|*`

Where:

`acc`	Is a keyword that begins a line for specifying ACC capabilities
`=|+=|-=|:=`	Are operators for adding, removing, or changing ACC capabilities
`capabilities`	Is a comma separated list of ACC capabilities
`module_names`	Is a comma separated list of module identifiers (or names). Specifying modules enables, disables, or changes (depending on the operator) the ability of the PLI function to use the ACC capability in all instances of the specified module.
`+`	Specifies adding, removing, or changing the ACC capabilities for not only the instances of the specified modules but also the instances hierarchically under the instances of the specified modules.
`%CELL`	Enables, disables, or changes (depending on the operator) the ability of the PLI function to use the ACC capability in all instances of module definitions compiled under the `` `celldefine `` compiler directive and all module definitions in Verilog library directories and library files (as specified with the `-y` and `-v` compile-time options.)
`%TASK`	Enables, disables, or changes (depending on the operator) the ability of the PLI function to use the ACC capability in all instances of module definitions that contain the user-defined system task or system function associated with the PLI functions.
`*`	Enables, disables, or changes (depending on the operator) the ability of the PLI function to use the ACC capability throughout the entire design. Using wildcard character could seriously impede the performance of VCS.

The operators in this syntax are as follows:

=	A shorthand for +=
+=	Specifies adding the listed capabilities that follow to the parts of the design that follow, as specified by module name, `%CELL`,`%TASK`, or `*` wildcard character.
-=	Specifies removing the listed capabilities that follow from the parts of the design that follow, as specified by module name, `%CELL`,`%TASK`, or `*` wildcard character.
:=	Specifies changing the ACC capabilities of the parts of the design that follow, as specified by module name, `%CELL`,`%TASK`, or `*` wildcard character, to only those in the list of capabilities on this specification. A specification with this operator can change the capabilities specified in a previous specification.

The ACC capabilities that you can specify for the functions in your PLI specifications are as follows:

`r or read`
To read the values of nets and registers in your design.

`rw or read_write`
To both read from and write to the values of nets and registers in your design.

`cbk or callback`
To be called when named objects (nets registers, ports) change value.

`cbka or callback_all`
To be called when named and unnamed objects (such as primitive terminals) change value.

`frc or force`
To force values on nets and registers.

`prx`
To set pulse error and pulse rejection percentages for module path delays.

`s or static_info`
The ability to access static information, such as instance or signal names and connectivity information. Signal values are not static information.

`tchk or timing_check_backannotation`
To back annotate timing check delay values.

`gate or gate_backannotation`
To back annotate delay values on gates.

`mp or module_path_backannotation`
To back annotate module path delays.

`mip or module_input_port_backannotation`
To back annotate delays on module input ports.

`mipb or module_input_port_bit_backannotation`
To back annotate delays on individual bits of module input ports.

The VCS Configuration File

You can use the configuration file to specify Radiant technology optimizations for parts of your design. The syntax of the statements in the configuration file is as follows:

```
module {list_of_module_identifiers}
{list_of_attributes};

instance {list_of_module_identifiers_and_
hierarchical_names} {list_of_attributes};

tree [(depth)] {list_of_module_identifiers}
{list_of_attributes};
```

Where:

`module`	Keyword that specifies that the attributes in this statement apply to all instances of the modules in the list, specified by module identifier.
`list_of_module_identifiers`	A comma separated list of module identifiers enclosed in curly braces: { }
`list_of_attributes`	A comma separated list of attributes enclosed in curly braces: { }
`instance`	Keyword that specifies that the attributes in this statement apply to: • All instances of the modules in the list specified by module identifier. • All the module instances in the list specified by their hierarchical names. • The individual signals in the list specified by their hierarchical names.
`list_of_module_identifiers_and_hierarchical_names`	A comma separated list of module identifiers and hierarchical names of module instances and signals enclosed in curly braces: { }
`tree`	Keyword that specifies that the attributes in this statement apply to all instances of the modules in the list, specified by module identifier, and also apply to all module instances hierarchically under these module instances.
`depth`	An integer that specifies how far down the module hierarchy, from the specified modules, you want to apply the attributes. You can specify a negative value. A negative value specifies descending to the leaf level and counting up levels of the hierarchy to apply these attributes. This specification is optional. Enclose this specification in parentheses: ()

The attributes for Radiant technology are as follows:

`noOpt`	Disables Radiant optimizations on the module instance or signal.
`noPortOpt`	Prevents port optimizations such as optimizing away unused ports on a module instance.

`Opt`	Enables all possible Radiant optimizations on the module instance or signal.
`PortOpt`	Enables port optimizations such as optimizing away unused ports on a module instance.

Enabling ACC Capabilities

In the PLT table entry format acc_spec is:

`acc` *`op capabilities`*`:`*`module_names`*`[+]`

Where:

`op` is one of

`=`	Shorthand for `+=`
`+=`	Add these capabilities to the specified scopes.
`-=`	Remove this capability from the specified scopes.
`:=`	Set exactly this capability to the specified scopes.

`capabilities` is one or more (comma separated) of

`r`	Read nets, registers and variables.
`rw`	Read as in r; write registers and variables.
`cbk`	Callbacks on named object value changes.
`cbka`	Callbacks on unnamed objects.
`tchk`	Backannotation of timing checks.
`gate`	Backannotation of gate delays.
`mp`	Backannotation of module paths.
`mip`	Backannotation of module input ports
`mipb`	Backannotation of individual bits of module input ports

You can specify one or more module in the table entry. ACC capabilities are enabled for module definitions, not module instances.

The + specifies enabling for the module instances hierarchically under these module definitions.

You can use the following for wildcard module instances:

`%CELL`	All modules tagged with `` `celldefine `` or extracted from libraries specified with the `-v` or `-y` options.
`%TASK`	All module definitions that contain task enabling statements or function calls for the task or function of this PLI table entry.
`*`	All module definitions.

CLI Commands

`.` `(period)`

Continue simulation.

`?`

Displays a list of the CLI commands and briefly describes what they do.

`alias [`*`alias_name`*` [`*`existing_command`*`]]`

Create or list command alias(es).

`always #`*`relative_time`*`|[@posedge|@negedge] `*`signal`

Set a repeating breakpoint. You can specify a simulation time interval, starting at the current simulation time and every time VCS runs the simulation for that interval, VCS halts simulation. You can specify a net or a register so that VCS halts when that signal changes value. You can also specify that the breakpoint only occur on a rising or falling edge of that signal.

`break #`*`relative_time`*`|[@posedge|@negedge] `*`signal`

Set a repeating breakpoint. This command is synonymous with the `always` command.

`continue`

Continue simulation. This command is synonymous with the **`.`** (period) command.

`delete `*`breakpoint_number`*

Delete a breakpoint. When you enter the `show break` CLI command, VCS lists the breakpoints that you have set with the `always` or `break` CLI commands. Each breakpoint in this list begins with a number indicating the order in which you set the current breakpoints.

You can use this `delete` command to delete or unset a breakpoint by specifying this number.

finish

Exit the simulation immediately. This command is the equivalent of VCS executing the `$finish` system task.

force *signal = value*

Force a signal to a value. Simulation events in your design, such as values propagating to a net or a procedural assignment statement assigning a value to a register, do not override this forced value.

help

Displays text-based online help listing the CLI commands and what they do. This command is synonymous with the `?` command.

info

Displays the current simulation time and the current scope.

line

Toggles line tracking. Line tracking is the display of the last line in your source code that VCS executed before simulation stopped. VCS displays the source file name and line number of the line in that file.

next

VCS executes the next line, and only the next line that it has scheduled for execution. You use this command to step through your code.

oformat %[b|c|t|f|e|g|d|h|x|m|o|s|v]

Sets the output format for displaying simulation values as described for format specifications in IEEE Std 1364-1995 pages 174-175. x is an alternative for hexadecimal.

once #*relative_time*|##*absolute_time* | [@posedge|@negedge] *signal*

Set a one shot breakpoint. A one shot breakpoint halts simulation only once.

print %[b|c|t|f|e|g|d|h|x|m|o|s|v] *signal*

Shows the current value of net or register in the specified format.

release *signal*

Releases a net or register from its forced value.

scope [*module_instance_hierarchical_name*]

Set or show scope. A scope in a location in the module hierarchy from which you can see and change values with the CLI.

set *reg_or_memory_address* [=]*value*
[,*reg_or_memory_address*[=]*value*]

Deposits a value on a register or a memory address. Simulation events in your design can override this value. This command is not valid for nets. The equal sign (=) operator is optional.

show [break|drivers|ports|scope|variables|?]

`break`
Lists, by number, the temporary and permanent breakpoints that are currently scheduled to halt simulation. You can use this number to delete or unset the breakpoint with the delete command.

`drivers` *net_or_reg*
Shows the value and strength of the net or register and for nets shows the line number in the source code of the statement that is the source of the value that propagated to this net.

`ports`
Shows the port identifiers of the instance that is the current scope and whether they are input, output, or inout ports, listed as `IN`, `OUT`, and `INOUT`.

`scope`
Shows the module instances in the current scope by their module identifier and module instance identifier.

`variables`
Shows the nets and registers declared in the current scope.

`?`
Displays this list or arguments to the `show` command and briefly describes what they do.

source *filename*

Executes CLI commands from a file.

tbreak [#*relative_time*| ##*absolute_time*|
@posedge| @negedge] *signal*

This command is synonymous with the `once` command.

trace

Line tracing displays the source file and line number of a statement before VCS executes the statement. This command toggles on and off line tracing. To use this command you must compile your design with the use the `-line` compile-time option.

unalias *alias_name*

Removes the alias.

`upscope`

Sets scope one level higher in the module hierarchy.