e Language Quick Reference

This card contains selected ***e*** constructs. For complete ***e*** syntax, see the *Specman e Language Reference*.

Abbreviations:

arg - argument	exp - expression	TCM - time-consuming method
bool - boolean	inst - instance	TE - temporal expression
enum - enumerated	num - number	

Predefined Types

bit	**byte**	**bool**	**longint**	**longuint**	**string**	**real**

int | **uint (bits:** *n* | **bytes:** *n* **)** *exp* **=** *exp*.**as_a(** *type* **)** // type conversion

list [**(key:** *field-name*)] **of** *type* **set**

Statements

import	**verilog ...**	**vhdl ...**	**type**	**struct**	**unit**
extend	**interface**	**template**	**numeric_type**		**routine**

User-Defined Types — Statements

struct *struct-type* [**like** *base-struct-type*] **{** *struct members* **};**

unit *unit-type* [**like** *base-unit-type*] **{** *unit members* **};**

Interface *interface-name* [**like** *base-interface-type*] **{** *interface members* **};**

type *type-name* **:** [**u**]**int (bits:** *n* | **bytes:** *n* **);**

type *enum-type***: [***name1***,** *name2***,** ...**];**

extend *type-name* **: [** *name* [=*n*], ... **];**

extend *struct-type* | *unit-type* **{** *additional struct or unit members* **};**

numeric_type *type-name* **:** *base-struct-name***;** // custom numeric type

Template Types

template (**struct** | **unit** | **interface**) *template-name* **of (***param-list***)**
[**like** *base-type*] **{***template members* **}:**

template-name **of (** *actual-param-list* **)**

Struct and Unit Members

fields	events	constraints	when conditions	cover groups
methods and TCMs			temporal struct|unit members	

Fields — Struct and Unit Members

[**static**][**const**][**!**][**%**]*field-name* : *type*; **list of** [**list of**...] *type*;

when *const-field* { ...}; *field-name*[*n*] **: list of** *type***;**

field-name **:** *unit-type* **is instance;**

Conditional Extensions using When — Struct and Unit Members

struct | **unit** *struct-type* | *unit-type* **{**
  field-name **:** *enum-type* | *bool-type* ;
  when *field-value* ... **{** *additional-members* **};**
};
extend *when-qualifier struct-or-unit-type* **{** ... **};**
where *when-qualifier* is:
  [*field-value*'] *field-name* for boolean types
  field-value['*field-name*] for enumerated values

Predefined Methods and Pseudo-Methods

check()	**copy()**	**do_print()**	**extract()**	**finalize()**	**init()**
get_printed_lines()	**quit()**	**run()**	**rerun()**		

get_unit()	**get_all_units()**	**get_enclosing_unit()**
set_unit()	**connect_ports()**	**check_generation()**
get_children()	**raise_objection()**	**drop_objection()**

Simple / Event / Buffer Ports — Struct and Unit Members

port-inst-name:[**list of**] [*direction*] **simple_port of** *element-type* **is instance**;

port-inst-name:[**list of**] [*direction*] **buffer_port of** *element-type* **is instance**;

event-port-field-name:[**list of**] [*direction*] **event_port is instance**;

keep [**soft**] *port-exp.attribute*() == *value;*

keep bind(*port-exp1, port-exp2***)**;

keep bind(*port-exp1,* **external | empty** | **undefined)**;

Method/TLM Interface Ports — Statements, Struct and Unit Members

port-inst-name: [**list of**] *direction* **method_port of** *method-type* **is instance**;

keep bind(*port-exp1, port-exp2***)**;

keep bind(*port-exp1,* **external | empty** | **undefined)**;

port-exp1.**connect(***port-exp2* |**empty | undefined)**;

port-inst-name : [**list of**] [*direction*] **interface_port of** *tlm-intf-type*
  [**using prefix=***prefix* | **using suffix=***suffix*] [**is instance**];

UVM Style Syntax - Instead of "direction interface_port of", use:
interface_port of **interface_export of**
interface_imp of

port1-exp.**connect(***port-exp2* | "*external_uvm_path*" | **empty | undefined**)

Constraints — Struct and Unit Members

keep [*name* **is** [**only**]] [**soft**] *constraint-definition*

keep soft *exp* **== select {** *weight* **:** *policy*; ... **};**

keep (*bool-exp* ? *exp1*: *exp2*) == *exp3*;

keep *bool-exp1* [**=>** | **or** | **and**] *bool-exp2***;**

keep *exp* **in** *list*; **keep** *field-name* **in** [*range*];

keep *list1*.**sum**(*exp1*) == *exp2* **keep** *list1*.**count**(*bool-exp*) == *exp*

keep *list1*.**all_different**(*exp*) **keep** *list1*.**has**(*bool-exp*)

keep for each (*item* **) in** *list* **{** [**soft**] *constraint-bool-exp*; ... **};**

keep (*exp1***,** *exp2***,** … **) in_table {** *table-row* **;** ... **};**

keep *field-name*.**hdl_path() ==** "*string*" **;**

keep *bool-exp1* [**=>** | **or** | **and**] *bool-exp2***;**

keep *exp1* [**==** | **!=** | **>** | **<** | **>=** | **<=**] *exp2***;**

keep *exp1* [**+** | **-** | **/** | ***** | **%** | **>>** | **<<** | **&** | **|** | **^**] *exp2* == *exp3*;

Generation On the Fly — Actions

gen *gen-item* [**keeping {** [**soft**] *constraint-bool-exp* **;** ... **}**];

do *field-name* [**keeping** {*constraint*,...}] //sequences

Generation with Procedural Code — Methods of Any Struct

pre_generate() is also {...} **post_generate() is also** {...}

Events and Temporal Struct and Unit Members

event *event-name* [**is** [**only**] *TE*] [**using** [**also**] *temporal-operators*];

static event *event-name***;**

emit [*struct-inst*.]*event-name***;**

on [*const-path*.]*event-name* **{** *action*; ... **}** ;

on [*const-path*.]*event-port***$ {** *action*; ... **}** ;

expect [*rule-name* **is** [**only**]] *TE*
  [**else dut_error(***string-exp***)**] [**using** [**also**] *temporal-operators*];

temporal-operators syntax: *operation condition*
abort | [**exclusive**_]**start** | **stop** @*event* | **none** | **empty**

Predefined Events

sys.any	*struct-inst*.**quit**	**sys.new_time**

Temporal Expressions (TEs)

Basic Temporal Expressions

@[*struct-inst*.]*event-name*	**change** | **fall** | **rise(***port***$)** **@sim**	
change | **fall** | **rise(***exp***)**	**true(***bool-exp***)**	**cycle**

Boolean Temporal Expressions

TE1 **and** *TE2*	*TE1* **or** *TE2*	**not** *TE*	**fail** *TE*

Complex Temporal Expressions

delay(*exp***)**	**{** *TE*; *TE*; ... **}**	**detach(***TE***)**
TE1 **=>** *TE2*	*TE* **exec {** *action*; ... **}**	**[** *n* **]** [***** *TE*]
TE **@**[*struct-inst*.]*event-name*		

Time-Consuming Actions

wait [[**until**] *TE*]; **sync** [*TE*];

Preprocessor Directives

#define [']*name* [*replacement*] **#undef** *name*

#if[n]**def** [']*name* **then** {*e-code*} [**#else** {*e-code*}] ;

Macros

define *<tag'syntactic-category>* **"***match-exp***"** **as** {*replacement*}

define *<tag'syntactic-category>* **"***match-exp***"** **as computed** {*action*;…}

Syntactic Categories

statement struct_member action exp type cover_item command

Tables

table [**count**] **{** *table-row* **;** … **} with {** *body* **};**

table [*set-literal* **] with {** *body* **};**

table from *table-operator*(*param-list*) [**using** *options*] **with** { ***body*** };

Annotations

[**repeatable**] **annotation** *@annotation-type-name* **{** *struct members* **};**

@annotation-type-name[(*attr-name*[=*attr-value*],…)] … *program-entity*

Variable Declarations and Assignments — Actions

var *var-name*: *type-name* = *exp*; **var** *var-name* := *exp*;

var-name = *exp*; [*struct-exp*.]*field-name* = *exp*;

Conditional Procedures — Actions

if *bool-exp* [**then**] **{** *action*; ... **}**
  [**else if** *bool-exp* [**then**] **{** *action*; ... **}**] [**else {** *action*; ... **}**] ;

case { *bool-exp*[:] **{** *action*; ... **} ;** [**default**[:] **{** *action*; ... **} ;**] **};**

case *case-exp* **{** *case-action-block;...* [**default**[:] **{** *action*; ... **} ;**] **};**

Checks — Actions

check [[name] **that**] *bool-exp* [**else dut_error(***message-exp*, ...**)**]

Methods and TCMs — Struct and Unit Members

[**static**] [**final**] *method-name* **(**[*param-list*]**)** [: *return-type*] [*@event*] **is**
  {*action*;...} // @event required for TCM
param-list syntax: *param-name*:*param-type*[=*default-exp*], ...

[**static**] *method-name* **(**[*param-list*]**)** [: *return-type*] [*@event-type*] **is**
  [**also|first|only**] {*action*;...}

return [*exp*];

Interface Methods — Interface Members

method-name **(**[*param-list*]**)** [**:** *return-type*]**;**

Invoking Methods and TCMs — Actions

[[*struct-exp*].]*method-name*([*param-list*])

[*struct-type*::]*static-method-name*([*param-list*])

start TCM() // starts TCM in a new thread

TCM2()@*event-name* **is** {*TCM1*(); *method*();};

method1() **is {** *method2*(); *method3*(); **};**

method() **is { start** *TCM*();};

Loops — Actions

for *i* **from** *exp* [**down**] **to** *exp* [**step** *exp*] [**do**] **{** *action*; ... **};**

for each [*struct-type*] (*list-item*) [**using index** (*index-name*)]
  in [**reverse**] *list* [**do**] **{** *action*; ... **};**

for each [*struct-type*] (*set-item*) **in_set** [**reverse**] *set* [**do**] **{** *action*; ... **};**

for each [**line**] [(*line-name*)] **in file** *file-name* [**do**] **{***action*; ... **};**

while *bool-exp* [**do**] **{** *action*; ... **};**

Ways to exit a loop: **break; continue;**

Operators

Operator precedence is left to right, top to bottom in the list

[ ]	list indexing	**[..]**	list slicing
[:]	bit slicing	*f***()**	method or routine call
.	field selection	**in**	list/set inclusion
{... ; ...}	list concatenation	**%{... , ...}**	bit concatenation
~	bitwise not	**!, not**	boolean not
+, -	unary positive, negative	***, /, %**	multiply, divide, modulus
+, -	plus, minus	**>>, <<**	shift right, shift left
<, <=, >, >=	comparison	**is** [**not**] **a**	subtype identification
==, != **list_ptr== !list_ptr=**	boolean equal, not equal	**===,!==**	Verilog 4-state compare
~, !~	string matching	**&, |, ^**	bitwise and, or, xor
&&, and	boolean and	**||, or**	boolean or
!, not	boolean not	**=>**	boolean implication

a **?** *b* **:** *c* conditional "if a then b, else c"

Sequences

sequence *seq-name* [**using** *sequence-option*,...];

Options: **item** = *item-type* // default: virtual sequence
created_driver = *driver-name* // default: seq_name_driver
created_kind = *kind-name* // pre-defined: MAIN, SIMPLE, RANDOM

body() @driver.**clock is** [**only**] { ... };

do *field-name* [**keeping** {*constraint*;...}]

do [*when-qualifiers*] *field-name* [**on** *driver-exp*] [**keeping** {*constraint*;...}]

Sequence-Driver API

gen_and_start_main: bool **event** *item-done*

bfm_interaction_mode: bfm_interaction_mode_t

arbitration_mode: seq_arbitration_mode_t

get_next_item(): item_type @clock

try_next_item(): item_type @clock

driver.**wait_for_grant(***seq*: any_sequence**) @sys.any**

driver.**deliver_item(***item*: any_sequence_item**)**

driver.**wait_for_item_done(***item*: any_sequence_item**)@sys.any**

driver.**execute_item(***item*: any_sequence_item**)**

Messages

message([*tag*], *verbosity*, *exp*, ***)*** [*action-block*]

Structured Debug Messages (SDMs)

msg_started([*tag*,]*verbosity*, *msg-id*, *data-struct***)** [**{***action-block***}**]

msg_ended([*tag*,]*verbosity*, *msg-id*, *data-struct***)** [**{***action-block***}**]

msg_transformed([*tag*,]*verbosity*, *msg-id*, *from-item, to-item***)** [**{***action-block***}**] // Reports transformation of existing data items

msg_changed([*tag*,]*verbosity*, *msg-id*, *new-state-desc***)**
[**{***action-block***}**] // Reports a significant event

msg_info([*tag*,]*verbosity*, *msg-id*, *item1*[, *item2*]**)** [**{***action-block***}**]
// Reports a significant event in the environment

Packing and Unpacking Pseudo-Methods

exp **= pack(** *pack-option*, *exp*, … **)**

unpack(*pack-option*, *value-exp*, *target-exp* [, *target-exp*, ...] **)**

Predefined Routines — Actions

Deep Copy and Compare Routines

deep_copy(*exp* : struct-type**)** : struct-type

deep_compare[**_physical**]**(***inst1*, *inst2*, *max-diffs***)**: list of string

Selected Configuration Routines

set_config(*category*, *option*, *option-value* **)**

get_config(*category*, *option* **);**

Selected Arithmetic Routines (arguments are integers)

min|**max (** *x*, *y***)**: int	**abs(***x***)**: int	**odd**|**even (***x***)**: bool
ipow(*x*, *y***)**: int	**isqrt(***x***)**: int	**div_round_up(***x*, *y***)**: int

Bitwise Routines

exp.**bitwise_and** | **or** | **xor** | **nand** | **nor** | **xnor(***exp*: int\|uint**)**: bit

Selected String Routines

appendf(*format*, *exp*, ...**)**: string	**append(***exp*, ...**)**: string
exp.**to_string()**: string	**bin**|**dec**|**hex(***exp*, ...**)**: string

str_join(*list*: list of string, *separator*: string**)**: string

str_match(*str*: string, *regular-exp*: string**)**: bool

Selected Operating System Interface Routines

system("*command***")**: int	**date_time()**: string

output_from("*command***")**: list of string

get_symbol(*UNIX-environment-variable*: string**)** : string

files.write_string_list(*file-name*: string, *list*: list of string**)**

Stopping a Test

stop_run();

Set Pseudo-Methods

Selected Set Methods

set1.**union(***set2***)**		*set1*.**intersect(***set2***)**
set.**min()**	*set*.**max()**	*set*.**get_range(***num***)**

List Pseudo-Methods

Selected List Actions

add[0]**(***list-item* : list-type**)**

clear()	**delete(***index* : int**)**
pop[0]**()** : list-type	**push**[0]**(***list-item* : list-type**)**

Selected List Expressions

size(): int	**top**[0]**()**: list-type	**exists(***index*: int**)**: bool
reverse(): list	**sort(***exp*: exp**)**: list	**is_empty()**: bool
sum(*expr*: int**)**: int	**count (***exp*: bool**)**: int	**has(***exp*: bool**)**: bool
is_a_permutation(*list*: list**)**: bool		**all(***expr*: bool**)**: list

first(*expr*: bool**)**: list-type	**last(***exp*: bool**)**: list-type
key(*key-expr* : expr**)** : list-item	**key_index(***key-exp*: exp**)**: int
max(*expr*: int**)**: list-type	**max_value(***exp*: int**)**: int | uint
min(*expr*: int**)**: list-type	**min_value(***exp*: int**)**: int | uint

all_indices(*exp*: bool**)**: list of int

swap(*small*: int, *large*: int**)**: list of bit

unique(*exp***)**: list	**all_different**(*exp*)

Coverage Groups and Items

Struct and Unit Members

```
cover cover-group [ using [also] cover-group-options ] is [empty] [also] {
      item item-name [: type = exp ] [ using [also] cover-item-options ];
      cross item-name1, item-name2, ... ;
      transition item-name;
};
```

Coverage Group Options

text = *string*	**weight =** *uint*	**no_collect**	**radix = DEC|HEX|BIN**
when = *bool-exp*		**per_unit_instance** [=*unit-type*]	

instance_no_collect = *bool-exp*

Coverage Item Options

text = *string*	**weight** = *uint*	**no_collect**
radix = DEC|HEX|BIN	**when =** *bool-exp*	**at_least** = *num*
per_instance = *bool*	**ignore** | **illegal =** *cover-item-bool-exp*	

instance_no_collect | **instance_ignore** |**instance_illegal =** *bool-exp*

ranges=range([*n*..*m* **],** *sub-bucket-name*,
sub-bucket-size, *at-least-number* **);**

cādence®

Specman Quick Reference

This card contains selected Specman commands and procedures. For more information, see the *Specman Command Reference.*

Abbreviations:	dir - directory	exp - expression
	inst - instance	num - number

General Help

cdnshelp | **sn_help.sh** // opens Cadence Help

Specman **help** command   **Help** button in GUI

Creating an HDL Stub File

write stubs -xmvhdl | **-xmsv** | **-ver**[**ilog**] | **-xmsc | -xmvlog| -esi** [*file-name*] // XLM

write stubs -ver[**ilog**] |**-qvh** |**-mti_sv** |**-osci** |**-vcs** |**-vcssv** |**-esi** [*file-name*]

Compiler Script

%sn_compile.sh // displays compiler script options

%sn_compile.sh top.e // creates an executable named "top" with compiled top.e module (and all other modules loaded by top.e)

%**sn_compile.sh** *e-module* **-elib** // creates an *e*-library

%sn_compile.sh -shlib -exe top.e // creates a shared library and executable that can be loaded dynamically into a simulator (example-. Modelsim)

%sn_compile.sh -sim vcs -vcs_flags "*file1.v ... specman.v*" *top.e*
// creates a Specman executable named "vcs_top" that includes VCS, compiled top.e, and Verilog source files

Some Common Switches

-sim // specifies name of the simulator to be linked
(xsim, xl, vcs, vcssv, xmvlog, xmvhdl, xmsim**)**

-enable_DAC // compiles define as computed macros, table operators, annotation types in the same compilation phase with usage

-shlib // creates a shared library

-parallel // improves performance by compiling modules in parallel

Starting Specman Standalone

%specman [**-p**[**re_commands**] *commands* | @*cmd-file*.ecom]
[**-c**[**ommands**] *commands*...] [**-e** | **-gui**]

Switching between Specman and Simulator Prompts

<Return> // switches from Specman to the simulator

sn [*spmn-cmd*] // switches from simulator to Specman

xm *xm-cmd* // passes simulator command from Specman to Xcelium

Starting Specman with a Simulator

%specrun [**-p**[**re_commands**] *commands* | @*cmd-file*.ecom]
[**-c**[**ommands**] *commands*...] [**-e** | **-gui**] **-dlib |** *linked-specman-executable-and-parameters*
// Specman invocation using a linked executable or dynamically linked to a shared library

Xcelium Simulator

%xrun *file1.v file2.v test.e* **-snprerun** "@batch.ecom" // compiles Verilog files and e file, and executes pre-commands)

ModelSim

vsim -c -keepstdout *top-module vsim-options*

QHSim:
% qhsim -c top_try

VCS

integrated-vcs-executable **-ucli** [*vcs-options*]

Selected xrun Options To Use with Specman

-defineall *macro* // defines macro for all compilers

-endsnstage // marks the end of a list of e files to be compiled into the same compilation unit

-snstage *stagename* // compiles all e files as a staged compile

-nosncomp // prevents compiling *e* input files

-snchecknames // generates warning for incorrect HDL paths

-snload *e-files* // loads *e* files before HDL access generation

-snprerun **"***commands***"** // executes commands before simulation

-snseed *seed* // passes seed to Specman

-snset *"commands"* // specifies commands to be executed before compiling or loading *e* files

-snshlib *shared-lib-path* // uses specified *e* precompiled shared lib

Syntax Examples:

% **xrun -snshlib libsn**_*e-module*.**so** *hdl-files e-module*
% **xrun -snstage** *stage-name e-files* **-snstage** *stage-name e-files* ...
-endsnstage *e-files hdl-files*

xrun Coverage Options

-covworkdir *dir*   **-covscope** *scope*   **-covtest** *test*

HAL e Linting Command

hal [**-check** | **-nocheck** *category*[**:***category*...]] [**-design_info** *info-file*]
[**-rulefile** *definitions-file*] [**-snshlib** *shared-lib-file*] [**-esv** *esv-file*] [*e-files*]

Categories:

ALL_E	E_COVERAGE	E_LINT	E_PERFORMANCE
E_STYLE	E_TOOL	UVM_E	

Specman: Main Configuration Options

Categories

run	**cover**	**gui**	**xcelium**
memory	**simulation**	**print**	**debugger**
gen			

config *category -option=value* // change configuration

show config [*category* [*-option*]]

write config [**to**] *file-name*

read config [**from**] *file-name*

Test Phase Commands

test | **setup** | **generate** | **start** | **run** [-*option* = *value*, ...] // options are the related configuration options.

check   **finalize**   **extract**

Saving and Restoring the State

load [**-check**] [**-if**] *e-files*

restore [**-retain** | **-noretain**] [*esv-file*]

reload [**-retain** | **-noretain**] [*esv-file*]

sav[**e**] [**-with_logs**] *esv-file*

set retain state [**-off**]

Coverage Commands

read cov[**er**[**age**]] [**-merge -file =** *merge-filename*] *wildcard-filename,...*

write cov[**er**[**age**]] [**-merge**] *filename*

clear cov[**er**[**age**]]

sh[**o**[**w**]] **cov**[**er**[**age**]] [**-kind** = **full** | **sum**[**mary**] | **spread**[**sheet**]] [**-f**[**ile**] = *file-name*] [**-contr**[**ibutors**] [= *num*]] [**-win**[**dow**]] [*struct-type*[.*group-name*[.*item-name*]]] [,...]

sh[**o**[**w**]] **cov**[**er**[**age**]] **def** [*struct-type*[.*group-name*[.*item-name*]]]

rank co[**ver**] [**-sort_only**] [**-recover**] [**-window**] [**-file**=*file_name*] [**-initial_list**=*file_name*] [*struct-type*[.*group-name*[.*item-name*]]]

Waveform-Related Commands

set wave [**-mode**=*working-mode*] *viewer*   // not needed for Xcelium

tra[**c**[**e**]] [**on**] **change -wave -event**[**s**]=*event_name* [-**event**[**s**]=*event_name* ...] *exp*

tra[**c**[**e**]] **events -wave** ([*struct-type*.*event-type* | *flag*])

Memory Commands

sh[**ow**] **mem**[**ory**] [*struct*] [**-re**[**cursive**]

sh[**ow**] **mem**[**ory**] [**-depth =** *unit-e-path*] [**-depth =** *num*]

sh[**ow**] **path** *struct* | **-type =** *type-name* | **-full**

Message Commands

set message *unit* [**-tags=***tags* | **all**] [**-screen**] [**-trans**] [**-file=***file*]

[**-verbosity=***verbosity*] [**-nonrec**]

set message *unit* **-off** [**-tags=***tags* | **all**] [**-screen**] [**-trans**] [**-file=***file*] [**-nonrec**]

set message *unit* **-format=***format* [**-tags=***tags* | **all**] [**-screen**] [**-file=***file*] [**-nonrec**]

set message -style=*style* [**-verbosity=***verbosity*] [**-tags=***tags*]

show message *unit* [**-tags=***tags* | **all**] [**-screen**] [**-trans**] [**-file=***file* | **all**] [**-rec**[**ursive**]] [**-full**]

Event Commands

sh[**ow**] **event**[**s**] [*time*[..[*time*]] [*struct.event*] // wildcards allowed for event commands

sh[**ow**] **event def**[**initions**] [*struct.event* [,…]]

collect event[**s**] [*struct.event* [,…]] [**on** | **off**]

trace event[**s**] [**-off** | *struct.event* | **-off**]

trace event[**s**] **-wave** [*struct.event* | **-off** | **-show** | **-help**]

del[**ete**] **event**[**s**]

Show Pack and Unpack Commands

show pack(*pack-option*, *exp*, ...**)**

show unpack(*pack-option*, *value-exp*, *target-exp1* [,*target-exp2*,...]**)**

Log Commands

set log *file-name* [**{***command;...***}**] **set log off**

Shell Commands

shell *shell-command*

Print and Report Commands

p[**r**[**int**]] *exp*[, …] [**using** *print-options*]

rep[**ort**] *list-exp*, **{**[*headers*]**}**, *exp*,… [**using** *print-options*]

Note: Use the **show config print** command to display print options.
Examples:

```
    print sys.packets using radix=HEX
    report sys.packets, {"Addr \t Indx"; "%d \t %d"},.address,index
```

tree [*struct* | *list-exp*] // display the contents of a struct or list

write doc [**-l**[**oad**]] [**-path=***path*][**-dir=***dir*] [**-overwrite**] [**-no_show**] [**-detail**] [**-public**] [**-protected**] [**-package**] [**-private**] [**-no_source_links**] *e_verification_package_name* | @modules, …
// generate a multi-file, hierarchical eDoc report

Sequence Debug Command

tra[**ce**] **seq**[**uence**] [*driver-e-path*] [**-v=** **verbosity** | **off**] [**-file =** *file, ...*] [**-screen**] [**-trans**]

Generation Debugger Commands

break [**on**] **gen** [**action** *id* [**cfs** *id*]] [**error**] [**field** *struct_name.field_name*]
// set generation break point; enable collection of generation information
Examples:
break on gen error // collect generation information and stop on next contradiction
break on gen field my_packet_s.* // collect generation information and stop on next generation of any field of my_packet_s

sh[**ow**] **gen** [**–instance** *instance-name*[.*fieldname*] | **-ascii**]

Source Code Debugger Commands

cont[**inue**] [**to** *breakpoint-syntax*] **step_any**[**where**]

st[**ep**] **ne**[**xt**] **fin**[**ish**] **abort**

Setting Breakpoints

b[**reak**] [**once**] [**on**] *break-option* [@*module*] [**if** *cond*]
lb[**reak**] [**once**] [**on**] *break-option* [@*module*] [#[*thread-handle*] [**if** *cond*]
Where *break-options* are:

- **c**[**all**] [**ext**[**ension**]] [*struct-wildcard*.]*method-wildcard*
- **re**[**urn**] [**ext**[**ension**]] [*struct-wildcard*.]*method-wildcard*
- **event** [[*struct-wildcard*.]*method-wildcard*]
- *special-event-type* [*special-wildcard*]

b[**reak**] [**once**] [**on**] **l**[**ine**] [*line-number*] [@*module* | @*expansion-index*] [**if** *cond*]

lb[**reak**] [**once**] [**on**] **l**[**ine**] [*line-number*] [@*module* | @*expansion-index*] [#[*thread-handle*] [**if** *cond*]

b[**reak**] [**once**] [**on**] **change** *exp* | **error** | **interrupt** | **sim** | **contention**

b[**reak**] [**on**] **alloc** [*memory-size*]

Managing Breakpoints

delete | **disable | enable break** [**last** | *id-number* | **"***pattern***"**]

show breakpoint

Setting and Managing Watches

[**l**]**watch** *exp* [**-radix = DEC | HEX | BIN**] [**-items =** *value*] [#*thread-id*]

customize watch *watch-id* [**radix = DEC | HEX | BIN**] [**-items =** *value* | **default**]

show watch **delete watch** [*watch-id*]

Setting Traces

tra[**ce**] [**once**] [**on**] *trace-option* [@*module-name*] [**if** *cond*]
ltra[**ce**] [**once**] [**on**] *trace-option* [@*module-name*] [#[*thread-handle*]] [**if** *cond*]
Where *trace-option* is:

- **c**[**all**] [**ext**[**ension**]] [*struct-wildcard*.]*method-wildcard*
- **re**[**urn**] [**ext**[**ension**]] [*struct-wildcard*.]*method-wildcard*
- **l**[**ine**] [*line-number*]
- *special-event* [*special-wildcard*]

tra[**ce**] [**once**] [**on**] **change** *exp* | **contention**

tra[**ce**] [**on**] **packing** | **reparse**

tra[**ce**] [**on**] **check** [*struct-wild-card.method-wild-card*] [@*module-name*]

tra[**ce**] **deep**

tra[**ce**] **glitch** [**on | off**] **c**[**all**] [*port-e-path*]

tra[**ce**] *internal-port-activity* [*unit-wildcard* | *port-wildcard*] [*destination*] [**off**]

tra[**ce**] *external-port-activity* [[*agent-wildcard*.]*unit-wildcard*. | *port-wildcard*] [*destination*] [**off**]

Special Events and Special Wildcards

Special Event Name	Special Wildcard
tcm_start	*struct-wild-card.tcm-wild-card*
tcm_end	*struct-wild-card.tcm-wild-card*
tcm_call	*struct-wild-card.tcm-wild-card*
tcm_return	*struct-wild-card.tcm-wild-card*
tcm_wait	*struct-wild-card.tcm-wild-card*
tcm_state	*struct-wild-card.tcm-wild-card*
call	*struct-wild-card.method-wild-card*
return	*struct-wild-card.method-wild-card*
sim_read	*signal-name-wild-card*
sim_write	*signal-name-wild-card*
output	*text wild-card*

Command-Line Mode Debugging Commands

sh[**ow**] **sta**[**ck**] // show the calls stack for the current thread

sh[**ow**] **thr**[**ead**] // show all threads

sh[**ow**] **thr**[**ead**] **so**[**urce**] [#[*thread-id*[.*call-id*]]]
// show the ***e*** source for the current thread

sh[**ow**] **thr**[**ead**] **tr**[**ee**] [#[*thread-id*]]
// show the full tree of calls for the current thread

sh[**o**[**w**]] **def**[**ine**[**s**]] [**-v**] [**-e**] [**"** [`]*wildcard-name***"**]
// -e : e defines only; -v : Verilog defines

sh[**ow**] **macro** [**-full**] [**-nest**] **-line=***line-no* @*module-name* | #*expansion*

sh[**ow**] **macro** [**-full**] [**-nest**] "*e-code-string*"
-macro = *macro-name-exp* | **-match_exp =** *macro-match-exp*

collect [**-file**=file-name] [**-after**=module-name] [**-reload**] *struct-name.method,…* // collect method extensions and print to log

sh[**o**][**w**] **mod**[**u**[**les**]] [**-checksum** | **-win**[**dow**]]

trace reparse // trace macro reparse during load/compile

trace tables // trace table expansion during load/compile