e Language Quick Reference

November 2011

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

Abbreviations:

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

Predefined Types

bit	**byte**	**int**
bool	**uint**	**string**
int | **uint (bits:** *n* | **bytes:** *n* **)**		**real**

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

exp **=** *exp*.**as_a(** *type* **)** // type conversion

User-Defined Types — Statements

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

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

Struct and Unit Members

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

Fields — Struct and Unit Members

[**const**][**!**][**%**]*field-name* : *type*;

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

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

list of [**list of**...] *type*;

Conditional Extensions using When — Struct and Unit Members

```
struct | unit struct-type | unit-type {
      field-name : enum-type;
      when name1 struct-type | unit-type { additional members };
};
extend name1 struct-type | unit-type { ... };
```

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* // “[only]” option in IntelliGen only

keep soft *bool-exp* **== select {** *weight* **:** *value*; ... **};**

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

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

keep *exp* **in** *list*;

keep *exp1* ? *exp2* : exp3;

keep *list1*.**all_different**(*exp*)

keep *list1*.**sum**(*exp1*) == *exp2*

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

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

Generation On the Fly — Actions

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

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

Predefined Methods for Any Struct

run()	**extract()**	**check()**	**finalize()** **visualize()**
init()	**pre_generate()**	**post_generate()**	**quit()**

Predefined Methods for Any Unit

get_unit() **get_all_units()**	**get_enclosing_unit()**
set_unit() **connect_ports()**	**check_generation()**
try_enclosing_unit()	**connect_pointers()**

Temporal Struct and Unit Members

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

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

expect | **assume** [*rule-name* **is** [**only**]] *TE*
    [**else dut_error(** “*string*”, *exp*, ... **)**];

Events

event *event-name* [**is** [**only**] *TE*];

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

Predefined Events

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

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

Time-Consuming Actions

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

Preprocessor Directives — Statements, Struct Members or Actions

#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

Variable Declarations and Assignments — Actions

var *var-name* **:** *type*;	**var** *var-name* **: =** *value*;
var-name **=** *exp* **;**	[*struct-exp*.]*field-name* <= *exp*

Template Types

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

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

Methods and TCMs — Struct and Unit Members

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

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

return [*exp*]

Invoking Methods and TCMs — Actions

[[*struct-exp*].]*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*();};

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*; ... **}** ;] **}**;

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 [**line**] [(*line-name*)] **in file** *file-name* [**do**] **{***action*; ... **}**;
while *bool-exp* [**do**] **{** *action*; ... **}**;

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

Checks — Actions

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

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** range list
{... ; ...} 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
==, != 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* // virtual sequence if not used
created_driver = *driver-name* // default: seq_name_driver
created_kind = *kind-name* // default: seq_name_kind
// pre-defined seq_name_kind: 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

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

event *item-done*

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*]

Message-Logger API

tags: list of message_tag

verbosity: message_verbosity
// NONE (default), LO, MEDIUM, HIGH, FULL

to_file: string
// target log file for printing (default extension is .elog)

to_screen: bool // TRUE by default

set_actions(*verbosity*: message_verbosity, *tags*: list of message_tag, *modules*: string, *text*: string, *op*: message_operation)

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*: struct-type, *inst2*: struct-type,
    max-diffs: int): list of string

Selected Configuration Routines

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

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

Selected Arithmetic Routines

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

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
reverse() : list	**sort(***exp* : exp) : list
sum(*expr* : int) : int	**count (***exp* : bool) : int
exists(*index* : int) : bool	**has(***exp* : bool) : bool
is_empty() : bool	**is_a_permutation(***list*: list) : bool
all(*expr* : bool) : list	**all_indices(***exp* : bool) : list of int
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
swap(*small* : int, *large* : int) : list of bit	
crc_8|32(*from-byte* : int, *num-bytes* : int) : int	
unique(*exp* : exp) : list	**all_different(***exp* : 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*]	

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*	

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

Specman
Quick Reference

January 2011

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

help [*string...* | "*reg-exp*"] // ASCII help	**sn_help.sh**
Help button in GUI	**cdnshelp** // help GUI

Creating an HDL Stub File

write stubs -ncvlog | **-ncvhdl** | **-ver**[**ilog**] | **-ncsc** | **-ncsv** | **-esi** [*file-name*] // IES only; stub files not required for irun

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

Example: specman -command "load top.e; write stubs -osci"

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* **-shlib –t** *tmp_directory*

%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 (xl, ncvlog, ncvhdl, ncsim, vcs, vcssv)

enable_DAC // compiles define as computed macros in the same compilation phase

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**]

Example:
specman -p "config print -radix = HEX" -p "load top" -e // starts Specman, sets print radix to hex, loads top.e, and enables command line editing mode.

Switching between Specman and Simulator Prompts

<Return> // switches from Specman to the simulator

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

nc *nc-cmd* // passes simulator command from Specman to IES

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

IES Simulator

%irun *file1.v file2.v test.e* **-snprerun** "@batch.ecom"// single call flow with IES (compiles Verilog files and e file, and executes pre-commands)

ModelSim

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

VCS

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

Specman-specific irun Options

-nosncomp // prevents compiling Specman input files

-sncheckname**s** // generates a warning if Specman references an incorrect HDL path

-sncompargs *strings* // passes arguments to sn_compile.sh

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

-snprerun "*comds*" // specifies Specman commands to be executed before simulation

-snseed *seed* // passes seed value to Specman

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

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

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

-defineall *macro* // defines macro from command line for all compilers

-intelligen // configures generator to use intelligen

Syntax Examples:

% irun –snshlib libsn_*e_module*.**so** *hdl_files e_module*
% irun –snstage *stage_name e_files* **-snstage** *stage_name e_files* **-snstage** … **-endsnstage** *e_files hdl_files*

irun Coverage Options

-covworkdir *work-dir* // selects coverage work directory

-covscope *scope-name* // selects coverage scope name

-covtest *test-name* // selects coverage test 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

***sav**[**e**] *options*

***res**[**t**[**ore**]] *options*

***rel**[**o**[**ad**]] *options* ***set retain state** *options*

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 IES

wave [**exp**] [**-when**] [**-depth**=*uint*] [**-field**[**s**] [**-event**[**s**] [**-thread**[**s**] *exp*

wave ev[**e**[**n**[**t**]]] [*struct-type*.*event-type*]

Memory Commands

***sh**[**o**[**w**]] **mem**[**ory**] *options*

***sh**[**o**[**w**]] **path** *options*

Log Commands

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

Message Command

***set me**[**s**[**sage**[**s**]]] options

Event Commands

show event[**s**] [*time*[..[*time*]] [*struct-name.event-name*] // wildcards allowed for event commands.

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

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

tra[**c**[**e**]] **ev**[**e**[**nt**[**s**]]] *options*

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

Show Pack and Unpack Commands

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

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

Shell Commands

shell *shell-command*

Specman: Main Configuration Options

Categories

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

***conf**[**ig**[**ure**]] *category -option=value...*

*sh[o[w]] **conf**[**ig**[**ure**]] // To see all configuration settings
*sh[o[w]] **conf**[**ig**[**ure**]] *category* // To see a specific category of settings
*sh[o[w]] **conf**[**ig**[**ure**]] *category -option …* // To see one or more specific options of a category

*write **conf**[**ig**[**ure**]]

*read **conf**[**ig**[**ure**]]

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** *options*

Sequence Debug Command

tra[**ce**] **seq**[**uence**] [*destination*] [**on** | **off**] [*wild*]
destination options: **msg**, **log**, **transaction**, **all** (default)

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**

In the next few sections, the *#thread-handle* option can only be used with the “l” (local) form of the command (e.g. **lbreak**, but not **break**). The special events and special wild cards used as options for some of the commands are listed separately at the end.

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 Wild Cards

Special Event Name	Special Wild Card
tcm_start	*struct-wild-card.tcm-wild-card*
tcm_end	*struct-wild-card.tcm-wild-card*
tcm_call	*struct-wild-card.tcm-wild-card*

Special Events and Special Wild Cards (continued)

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**[**_call**][**s**] *options*

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**]]

NOTE

*** --The command has '-h[elp]' option that prints the description of this command and its options. The parameters of this command can be given in any order.**

cādence