8.1 Naming convention guidelines

A naming convention provides consistency in a design project for the clarification of how objects in the design are used and classified. Users may not agree with our proposed naming convention, and may prefer an optional alternative. Our position is that as long as a consistent naming convention is explained and used across a project, the choice of the convention is not critical.

8.1.1 File naming

1. A file consists of three parts: a) Name that characterizes the function or level of the design or testbench, b) a type that characterizes its usage type (e.g., package, rtl, etc), c) an extension that characterizes the language (e.g., SystemVerilog, e, c). The format of a file typically adopted in industry is: *name_suffix.extension* In some cases, the format may include two or more suffixes, i.e., *name_suffix1_suffix2.extension.*
2. Each design name should include a suffix that describes the usage. The file name should match the design name. This approach helps to clarify the content and intent of a file. See Table 8.1.1-1.

Table 8.1.1-1 Design unit naming convention

Type of file	Suffix	Example	File name
Package	_pkg	`package cpu_pkg;`	cpu_pkg.sv
RTL	_rtl	`module cpu_rtl (…);`	cpu_rtl.sv
Behavior	_beh	`module cpu_beh(…);`	cpu_beh.sv
Properties	_props	`module cpu_props(..);`	cpu_props.sv
Testbench	_tb	`module top_tb();`	top_tb.sv
Checker (SV)	_chk	`checker cpu_chk();` *// SystemVerilog checker (See 5.0)*	cpu_chk.sv
Interface	_if	`interface usb_bus_if (..);`	usb_bus_if.sv
Program	_pgm	`program test_pgm;`	test_pgm.sv
Library	_lib	`package cpu_lib_pkg;`	cpu_lib_pkg.sv
Configuration	_config	`class counter_config;`	counter_config.sv
Driver	_driver	`class counter_driver;`	counter_driver.sv
Environment	_env	`class counter_env;`	counter_env.sv
Agent	_agent	`class counter_agent;`	counter_agent.sv
Monitor	_monitor	`class counter_monitor;`	counter_env.sv
Sequencer	_sequencer	`class counter_sequencer;`	counter_env.sv
Sequence	_sequence	`class counter_sequence;`	counter_env.sv
Transaction	_xactn	`class counter_xactn;`	counter_env.sv
Checker (TB)	_checker _verif	`class counter_checker;` *// verifier or checker in functionality*	counter_checker.sv
Test	_test	`class counter_test;`	counter_env.sv
Base	_base	`class counter_base_test;` *// base class*	counter_env.sv

3. Files that get directly compiled should have the **.sv** extension. All the files that are **`included** into a **.sv** file somewhere should have a **.svh** extension. This enhances the understanding on how SystemVerilog files are used.

4. Within a design, suffixes are used to characterize the type of the object or its active polarity. Table 8.1.1-2 summarizes an object naming convention.

Table 8.1.1-2 Object naming convention

Type of object	Suffix	Example

Type	_t
	_e
	_ev
		`typedef logic [WIDTH-1 : 0] word_t;`
		`typedef enum {OFF, RED, YELLOW, GREEN} lights_e;`
		`event clk_ev`

Active low variable	_n	`logic reset_n; // active low reset`

Constant and parameter	Upper Case	`parameter WIDTH=16;`
		`localparam DEPTH=256;`
		`module memory #( DEPTH, WIDTH) (..);`

modport	_mp	`drvr_if_mp`
Clocking block	_cb	`driver_cb`

8.1.2 Naming of assertion constructs

A naming convention for the assertions helps to identify the constructs, and becomes meaningful during the display of failed assertions or the access of instantiated assertions using the AVA Application Programming Interface (API). Table 8.1.2 provides a summary of recommended prefix notation. Note that the underscore '_' character is optional, but recommended when the first letter of the object name is in lower case, as it enhances readability.

Table 8.1.1.2 Recommended prefix named notation for assertion constructs

Type of object	Pre-fix	Example

se**q**uence	q	`sequence q_req;`
		`  $rose(ready) ##[0:4] req;`
		`endsequence : q_req`

property	p	`property p_reqack;`
		`  $rose(req) |=> ack;`
		`endproperty : p_reqack`

variable	v	`property p_req;`
		`  logic [31:0] v_data;  // local variable for data`
		`  logic [1:0] v_vie; // local variable for vie`
		`  ($rose(req), v_data=data, v_vie=vie) |=>`
		`  ack && buff=v_data && vie!=v_vie;`
		`endproperty : p_req`

local **v**ariable formal argument	lv	`sequence q_ab(input addr,  // formal argument`
		`  local inout int lv_addr); // local variable formal argument`
		`  (a, lv_address=addr) ##1 addr== lv_addr +1'b2;`
		`endsequence : q_ab`

assert*	a	`ap_reqack : assert property  (@ (posedge clk) p_reqack);`
cover*	c	`cq_req : cover sequence  ( @ (posedge clk) q_req);` `cp_reqack : cover property  (@ (posedge clk) p_reqack);`
assu**m**e*	m	`mp_reqack : assume property  (@ (posedge clk) p_reqack);` `mq_req : assume property  ( @ (posedge clk) q_req);`
restrict*	r	`rp_fn_mode : restrict property (@(posedge clk) scan_en == 0);`

* The prefix "a", "c", "m", is followed by the name of the property or sequence.