Naming Conventions for SystemVerilog Design and SystemVerilog Assertions

A Comparative Reference for the Cohen-Chibani RAG-SVA Framework

Ben Cohen, Kais Chibani
Cohen Publishing · systemverilog.us

1. Introduction

Consistent naming is what makes a large SystemVerilog Assertion (SVA) codebase debuggable, reviewable, and reusable across projects. This paper collects and compares naming conventions for SystemVerilog design and SVA from four sources: the Cohen-Chibani RAG-SVA framework's own property-naming rule; a convention Ben Cohen posted publicly on Verification Academy years before RAG-SVA existed; the lowRISC “Comportable” style guide, a widely cited public SystemVerilog style guide also followed by the OpenTitan project (a collaboration that includes Google's silicon team); and the Doulos “Easier UVM” coding guidelines. Each convention is illustrated with the same running example — a request/acknowledge handshake checker — so the different schemes can be compared directly rather than in the abstract.

The running example throughout this paper is the checker used elsewhere in the Cohen-Chibani RAG-SVA testbench-style examples:

```
checker chkr (input bit clk, resetn, req, rdy, ack);
  ap_req_ack: assert property (@(posedge clk) disable iff (!resetn)
    req |-> (rdy, ##[1:5] ack))
    else `uvm_error("SVA", $sformatf(
      "Error in req->ack. req=%b rdy=%b ack=%b", req, rdy, ack));
endchecker: chkr
```

2. Properties, Assertions, Covers, and Assumptions in the Cohen-Chibani RAG-SVA

The Cohen-Chibani RAG-SVA's own convention (PropertyNamingQuickRef.md) requires a **p_** prefix on every named property, and an **ap_** prefix on the assert label that instantiates it:

```
property p_req_to_ack_timing;
  @(posedge clk) disable iff (!resetn)
  req |-> (rdy, ##[1:5] ack);
endproperty

ap_req_to_ack_timing: assert property (p_req_to_ack_timing);
```

This mirrors a convention Ben Cohen posted publicly on Verification Academy, extended with a **cp_** label for the corresponding cover statement:

```
property region0_nonsec_check_mpu16;
  @(posedge clk) disable iff (...)
  ...
endproperty

ap_region0_nonsec_check_mpu16: assert property(region0_nonsec_check_mpu16);
cp_region0_nonsec_check_mpu16: cover property(region0_nonsec_check_mpu16);
```

Convention: *property names take a* ***p_*** *prefix; the assert label that instantiates a property takes* ***ap_****; the cover label that instantiates the same property takes* ***cp_****. The suffix after the prefix (****req_to_ack_timing****) stays identical across all three, so the property/assert/cover triad is recognizable at a glance and greppable as a unit.*

2.1 Assumptions: the mp_ Convention

The Cohen-Chibani RAG-SVA framework already documents an assumption-naming convention in DoThisFirst.md and RAG_RULES.md: an assume property statement is labeled with an **mp_** prefix, is required to carry a structured documentation header, and must be paired with a mandatory validation cover using the matching **cp_** label plus a _hit suffix:

```
/*
ASSUMPTION:    <Human-readable intent - plain English>
TYPE:          <Input Constraint | Output Focus | State Reduction | Temporal
                | Fairness/Blocking-Signal | Fairness/Activation>
JUSTIFICATION: <Spec reference OR reasoning>
RISK LEVEL:    <Low | Medium | High>
*/
mp_<descriptive_name>: assume property (@(posedge clk)
  disable iff (!reset_n)
  <antecedent> |-> <consequent>
);

// MANDATORY VALIDATION COVER
cp_<descriptive_name>_hit: cover property (@(posedge clk)
  disable iff (!reset_n)
  <cover_sequence>
);
```

A worked example, a Fairness/Activation assumption ensuring req[0] cannot stay low forever (which would make every downstream property vacuously true):

```
/*
  ASSUMPTION:    req[0] eventually asserts.
  TYPE:          Fairness/Activation
  JUSTIFICATION: L3-B — req[0] held low forever makes every downstream
                 property vacuously true. Section 4.5 requires fairness.
  RISK LEVEL:    Low
  */
  mp_fair_req0: assume property (@(posedge clk) disable iff (!rst_n)
    s_eventually req[0]);
```

Convention: *the TYPE field is drawn from a fixed enumeration (Input Constraint, Output Focus, State Reduction, Temporal, Fairness/Blocking-Signal, Fairness/Activation); the Fairness/* types are permitted only on mp_fair_* or mp_livelock_* assumptions and must be paired with a liveness-class assertion, never introduced into a purely safety-class property set. Together with* ***p_/ap_/cp_****, this gives the Cohen-Chibani RAG-SVA a complete four-way convention — property, assertion, cover, and assumption — each carrying its own prefix on a shared descriptive base name.*

3. RTL Signal and Port Naming (lowRISC "Comportable" Style Guide)

Source: github.com/lowRISC/style-guides. This guide governs the signals a checker or property observes, not the property names themselves — but the two must agree for a checker to read naturally.

- Clocks: **clk**; a design with multiple clocks uses a **clk_<domain>** prefix for the non-primary clocks
- Reset: **resetn / rst_n** — active-low, asynchronous by default
- Port direction suffixes: **_i** for inputs, **_o** for outputs, **_io** for bidirectional
- Register pairs: **_d** (next-state) and **_q** (registered); pipelined stages use **_q2, _q3**, etc.
- Active-low signals: **_n** suffix; differential pairs: **_p / _n**
- General case: **lower_snake_case** for signals and instances; **UpperCamelCase** for tunable parameters; ALL_CAPS for `define macros and true constants

Applied to the running example, the DUT-side ports behind the chkr checker would read:

```
module handshake_dut (
  input  bit clk_i,
  input  bit rst_ni,
  input  bit req_i,
  output bit rdy_o,
  output bit ack_o
);
```

4. SVA Naming in OpenTitan (lowRISC / Google)

Source: opentitan.org/book/hw/formal. OpenTitan documents a naming convention distinct from, and more specific than, a generic "**UpperCamelCase**" rule: every named construct takes an **UpperCamelCase** descriptive name plus a mandatory role suffix.

Suffix	Construct
A	Assertion
M	Assumption
P	Property
S	Sequence

OpenTitan wraps assertions in macros from prim_assert.sv rather than writing raw assert property statements, so the name/clock/reset boilerplate stays consistent project-wide:

```
`ASSERT(name, prop, clk, rst)              // generic concurrent assertion
`ASSERT_INIT(name, prop)                   // inside an initial block
`ASSERT_FINAL(name, prop)                  // inside a final block
`ASSERT_NEVER(name, prop, clk, rst)        // property must never occur
`ASSERT_KNOWN(name, signal, clk, rst)      // signal must never be X after reset
`ASSERT_AT_RESET(name, prop, rst)          // checked just before reset
`COVER(name, prop, clk, rst)               // coverpoint, same syntax as assert
```

A documented example, taken directly from the OpenTitan formal-verification reference, shows the **_A** and **_M** suffixes in use together with a symbolic variable:

```
`ASSUME_FPV(IsrcRange_M, src_sel >= 0 && src_sel < N_SOURCE, clk_i, !rst_ni)
`ASSERT(LevelTriggeredIp_A, $rose(rv_plic.ip[src_sel]) |->
       $past(rv_plic.le[src_sel]) || $past(intr_src_i[src_sel]), clk_i, !rst_ni)
```

Note: *OpenTitan's own reference material illustrates* ***_A*** *and* ***_M*** *with worked examples but does not publish a worked* ***_P*** *or* ***_S*** *example — its ASSERT/ASSUME_FPV macros take a property expression inline rather than requiring a separately declared property or sequence block. The example below applies the documented suffix rule to a named property and sequence by analogy, following the same request/acknowledge handshake as the running example; it is a construction for illustration, not a verbatim OpenTitan snippet.*

```
sequence AckWithinWindow_S;
  ##[1:5] ack;
endsequence

property ReqImpliesAck_P;
  @(posedge clk_i) disable iff (!rst_ni)
  req |-> AckWithinWindow_S;
endproperty

`ASSERT(ReqImpliesAck_A, ReqImpliesAck_P, clk_i, !rst_ni)
```

4.1 Security-Critical Assertions

OpenTitan's security IP blocks add a further, project-specific layer on top of the **_A/_M/_P/_S** scheme: macros defined in **prim_assert_sec_cm.svh** (for example, `ASSERT_PRIM_FSM_TRIGGER_ALERT) always generate assertions carrying the common prefix **FpvSecCm**, so the formal-verification tooling can isolate and batch-analyze fault-injection checks as a group, separately from ordinary functional assertions.

4.2 Action Blocks and Error Reporting in OpenTitan Macros

OpenTitan centralizes assertion action blocks rather than having each assertion hand-write its own else clause. Every **`ASSERT / `ASSERT_KNOWN / `ASSERT_NEVER** / etc. macro expands to a full **assert property (...) else `ASSERT_ERROR(name)** — the action block itself is delegated to one shared macro, defined in **prim_assert.sv:**

```
`define ASSERT_ERROR(__name) \
`ifdef UVM \
  uvm_pkg::uvm_report_error("ASSERT FAILED", `PRIM_STRINGIFY(__name), uvm_pkg::UVM_NONE, \
      `__FILE__, `__LINE__, "", 1); \
`else \
  $error("%0t: (%0s:%0d) [%m] [ASSERT FAILED] %0s", $time, `__FILE__, `__LINE__, \
      `PRIM_STRINGIFY(__name)); \
`endif
```

The same **`ASSERT(name, prop, clk, rst)** call therefore compiles two different ways depending on context:

- Under UVM (`ifdef UVM): fires a **uvm_report_error** with severity UVM_NONE (always printed regardless of verbosity), tagged "**ASSERT FAILED**", carrying the file and line where the assertion is declared.
- Outside UVM: falls back to a plain **$error** with a timestamp, file/line, and hierarchical path (%m).

This is the same "auto-detect UVM, pick the right reporting macro" behavior that the Cohen-Chibani RAG-SVA framework's own Item 19 setting governs — OpenTitan simply hard-codes the choice into one shared macro rather than exposing it as a per-session setting.

4.2.1 What You Gain, and What You Give Up

The macro-centralized action block does not support a custom error message. Unlike a hand-written **assert property (...) else $error("my custom text: req=%b", req)**, **`ASSERT** only ever prints the assertion's name, file, and line — never a custom string. OpenTitan's own documentation states this directly: the macro "*doesn't support a custom error message... the macro will print out the property name and the entire property code.*"

To get a custom message, the designer drops out of the macro and writes the raw assert property with its own else block, the way OpenTitan's immediate-assertion example does:

```
initial begin
  checkFifoWidth: assert (FifoDepth > 0) else begin
    $error("FifoDepth parameter should be > 0");
  end
end
```

This is a deliberate trade-off: consistency and low boilerplate for the common case, at the cost of message customization — accepted because the macro always reports which assertion fired and where, which is normally sufficient to locate it in the RTL.

4.2.2 Related Action-Block Variants

- **`ASSERT_STATIC_LINT_ERROR(name, prop)** — not a simulation action block at all; an elaboration-time mechanism that assigns an out-of-range **localparam** value to force a lint or synthesis error if prop is false, for checks that must be caught before tapeout rather than in simulation. See http://systemverilog.us/ASSERT_STATIC_LINT_ERROR_Trick.pdf for a discussion on
  The `ASSERT_STATIC_LINT_ERROR Macro
  A Deliberate Width-Mismatch Trick for Elaboration-Time Checks, and Its Risks

- A historical variant, **`ASSERT_RPT**, seen in earlier revisions of **prim_assert.sv**, performed the same **UVM / $error** branching through an assert_rpt_pkg package function rather than inlining the **`ifdef UVM** branch directly — functionally equivalent to the current **`ASSERT_ERROR**, just refactored since.

5. UVM Component and Class Naming (Doulos "Easier UVM" Guidelines)

Source: doulos.com, "Easier UVM Coding Guidelines." These conventions govern the class-based verification environment that would drive the checker above once it moves from a standalone testbench to a UVM-integrated one.

- Class names: lower_snake_case with a type suffix — **_env, _agent, _test**
- Standard instance names inside every agent: **m_sequencer, m_driver, m_monitor**
- Class member variables (not block-local variables) take an **m_** prefix
- Other established suffixes: _config (configuration objects), **_port / _export** (TLM connections), **_vif** (virtual interface handles)

Applied to the running example's UVM-integrated stage:

```
class req_driver extends uvm_driver #(req_item);
  `uvm_component_utils(req_driver)
  virtual dut_if m_vif;
  ...
endclass: req_driver
```

```
class req_env extends uvm_env;
  `uvm_component_utils(req_env)
  uvm_sequencer #(req_item) m_sequencer;
  req_driver                m_driver;
  ...
endclass: req_env
```

6. Side-by-Side Comparison

The Cohen-Chibani RAG-SVA convention and OpenTitan's convention both exist to make an assertion's role legible from its name alone, but they encode that role in opposite positions:

Aspect	Cohen-Chibani RAG-SVA	OpenTitan
Role marker position	Prefix on the label **(p_, ap_, cp_, mp_)**	Suffix on the name **(_A, _M, _P, _S)**
Property vs. assert distinguished?	Yes — **p_** vs **ap_**, same base name	Yes — **_P** vs **_A**, same base name
Cover distinguished?	Yes — **cp_**	No dedicated suffix; **COVER()** takes any descriptive name
Assumption distinguished?	Yes — **mp_**, with a mandatory documentation header and a paired **cp_<name>_hit** cover	Yes — **_M**
Delivery mechanism	Hand-written **property/assert/cover/assume** blocks	Macro-wrapped **(`ASSERT, `ASSUME_FPV, `COVER**, etc.)

References

Cohen,***Knowledge-Driven AI for SystemVerilog Assertions: A Verification Framework of Methodologies, Processes, and Coding Rules 2026*** RAG-SVA rules include PropertyNamingQuickRef.md, DoThisFirst.md, RAG_RULES.md

Cohen, B. Verification Academy forum post, “Cover property issue,” verificationacademy.com/forums/t/cover-property-issue/32860.

lowRISC. “Verilog Coding Style Guide,” github.com/lowRISC/style-guides.

lowRISC / OpenTitan. “OpenTitan Assertions,” opentitan.org/book/hw/formal.

Doulos. “Easier UVM Coding Guidelines,” doulos.com/knowhow/systemverilog/uvm/easier-uvm.