Gen-SVA: Source-Agnostic Generation of Formal-Ready SystemVerilog Assertions

Rules and Styles for Verification-Grade Output Regardless of Input Medium

Ben Cohen & Dr. Kais Chibani

Abstract — The generation of SystemVerilog Assertions (SVA) is increasingly automated, but the input is no longer only natural-language specifications. Assertions are now derived from English prose, SystemC and C/C++ reference models, RTL source, waveform images, and protocol tables. We argue two things. First, the **source medium is largely irrelevant** to assertion quality; what determines whether generated SVA is usable in a formal-verification flow is the set of **rules and styles** applied during generation. Second, when more than one representation of the same requirement is available, **using them together produces better assertions** than any single source alone — each medium covers the others' blind spots, turning cross-representation agreement into a built-in correctness check. This paper defines "Gen-SVA" [i]: a source-agnostic discipline that, whatever the input, converges on a single class of formal-ready output and exploits redundant representations when they exist. We enumerate the rules — intention classification, bounded operators, liveness conversion, fairness and anti-livelock infrastructure, mandatory coverage, and deterministic configuration — that make generated assertions provable rather than merely compilable.

1. The Problem Is Not the Source — It Is the Style

A persistent misconception in assertion automation is that the hard part is parsing the input. In practice, extracting candidate properties from English, SystemC, or RTL is the tractable half of the problem. The difficult half is ensuring that the resulting assertions are sound for formal verification. An assertion that compiles cleanly can still stall a formal engine, pass vacuously, or silently over-constrain the proof space — masking the very bugs it was written to catch.

This reframes the goal of any assertion generator. A generator that ingests a waveform image and one that ingests a C reference model face **the same downstream obligation**: the SVA they emit must obey the structural rules of formal verification. Gen-SVA is the name we give to that obligation — a discipline defined not by its front-end parser but by the rule set it enforces on the output. The front-end may change with every project; the rules must not.

2. Source-Agnostic Ingestion, Convergent Output

Source-Agnostic Ingestion, Convergent Output Different input media expose different classes of information, and each is blind to others. The Gen-SVA position is that all of them must be normalized into a common intermediate before properties are written:

- **English / natural-language specs** — carry intent and rationale but lack cycle-exact timing; ambiguity must be resolved before generation, not silently filled.
- **SystemC / C/C++ reference models** — carry algorithmic behavior and transaction ordering but hide clock-level timing and signal direction; the generator must infer cycle semantics explicitly.
- **RTL SV/VHDL source** — carries exact port maps and signal direction but hides design intent; structural extraction must be paired with a human-confirmed statement of what each property is meant to prove.
- **Legacy PSL/SVA** — carries pre-existing temporal properties with real historical verification intent, but predates the active rule corpus; banned operators, absent DOC9/INTENTION[ii] discipline, and undetected degeneracies may be embedded in code that has run in regression for years. Legacy properties must be re-validated against the current rule set before reuse, not imported as pre-verified ground truth — a property that never failed is not the same as a property that was correctly specified.
- **Waveform images / timing diagrams** — carry concrete cycle relationships but only for sampled scenarios; the generator must generalize a witnessed trace into a property without over-fitting to the example.
- **Protocol tables / register maps** — carry field-level structure but no temporal behavior; timing bounds must be supplied separately.

Whatever the source, the generation step must produce the same artifact: a signal-intention table plus a set of properties expressed in a restricted, formal-safe subset of SVA. The convergence is the point — a correct Gen-SVA flow yields equivalent assertions whether the requirement arrived as a sentence, a C function, or a picture of a waveform.

3. Multiple Representations Produce Better Assertions

Source-agnosticism says any single medium can drive generation. The stronger and more practical claim is this: when the same requirement is available in more than one representation, feeding them **together** yields better assertions than any one of them in isolation. The reason follows directly from the fact that each medium is blind to a different class of detail. English carries intent but not timing; a C model carries algorithm but not signal direction; a waveform carries exact cycle relationships but only for sampled cases; RTL carries port-exact structure but not rationale. Each representation is therefore a partial projection of the true requirement — and the projections are complementary.

Combining them produces three concrete benefits:

- **Gap-filling** — a missing latency bound in the English spec can be recovered from a waveform; a missing signal direction in a C model can be recovered from the RTL port map. The union of representations is more complete than any member.
- **Cross-validation** — when two representations agree on a property, confidence rises; when they disagree, the conflict surfaces a real specification ambiguity that would otherwise have been silently resolved by a guess. Disagreement is a feature: it is an early bug, caught before code.
- **Tighter bounds and intent** — a waveform pins down concrete timing that prose leaves vague, while prose supplies the design intent that keeps a witnessed trace from being over-fit into an accidental, too-rigid property.

This is the generative counterpart of the representational translation loop: rather than translating one format into another, Gen-SVA cross-references all available formats during generation and reconciles them into a single intention table. The practical guidance is simple — **if you have the requirement in more than one form, supply all of them**. The generator should treat agreement as corroboration and disagreement as a mandatory clarification gate, never discarding a representation merely because another was sufficient to proceed.

4. The Intention Boundary: The First Rule of Formal-Ready SVA

The single most consequential rule in Gen-SVA is the classification of every signal by its true driving source, not by superficial port direction. Environment-driven signals become `assume` constraints; design-driven outputs and tracked states become `obligation` targets verified by assertions. Violating this boundary is the most dangerous error an automated generator can make:

```
// CRITICAL OVER-CONSTRAINT VIOLATION
assume property (gnt |-> ##1 !gnt);   // gnt is a DUT OUTPUT
```

Because **gnt** is produced by the design, placing it inside an **`assume`** instructs the formal engine to constrain the design's own outputs. The tool prunes every state in which the design would have violated the rule and reports a false `PROVEN`, concealing real bugs. A source-agnostic generator is especially exposed: an image or a C model does not announce signal direction, so the generator must derive it and a human must confirm it. Gen-SVA mandates a PORT-INTENT table as a hard gate before any property is emitted — the same gate whether the input was English or RTL.

5. Operator Discipline: A Formal-Safe SVA Subset

Not all legal SVA is safe for formal tools. Constructs that are convenient in simulation cause state-space explosion, non-convergence, or vacuous passes in formal proof. Gen-SVA restricts generation to a bounded, formal-safe subset regardless of how idiomatically the source might suggest otherwise:

- Banned: unbounded delays and repeats (`##[1:$]` without `strong()`, `##[0:$]`), and simulation-centric operators (`until`, `within`, `firstmatch`) that explode or mislead the proof.
- Banned: non-overlapping implication (`|=>`), standardized instead on the overlapping form `|-> ##1` to remove cycle-interpretation ambiguity.

- Required: finite, parameter-driven windows (`##[1:N]`) with every bound traceable to a stated latency contract.

This subset is non-negotiable and source-independent. A waveform that shows a response “eventually” and a sentence that says “must eventually” are both lowered into the same bounded construct — avoiding operators that may challenge the formal engine's convergence. FV tools can handle ‘$’ sometimes; the convergence depends on property/design’s complexity.

6. Liveness, Fairness, and Anti-Livelock

Liveness is where naive generators most often produce green-but-meaningless results. A claim such as “every request must eventually receive a grant” passes vacuously if the tool can model a trace where the request never fires, or where an input is held in a blocking state forever. Gen-SVA handles liveness in three coupled moves, applied whenever the source expresses an “eventually” intent in any medium:

- **Bounded conversion** — unbounded liveness language is lowered to a finite form `trigger |-> ##[1:MAX] done`; an unbounded pure-liveness form using `strong()` is used only when no meaningful bound exists.
- **Activation fairness** — for every liveness property the full intention table is scanned for any input that could block progress, and a fairness assumption is generated so the tool cannot hold that signal in its blocking state forever.
- **Anti-livelock** — a continuously toggling input that could restart the antecedent indefinitely is constrained so the in-flight transaction is allowed to complete.

These companion constraints are generated automatically from the signal table, not from the surface text of the property. That is what makes them source-agnostic: the obligation to add fairness comes from the classification of the signals, which exists no matter how the requirement was originally expressed.

7. Anti-Vacuity, Coverage, and Traceability

A generated assertion is not trustworthy until it is shown to be reachable and its assumptions shown to be non-blocking. Gen-SVA enforces a documentation and coverage discipline as part of generation, not as an afterthought:

- **Coverage pairing** — every assumption is paired with a hit-coverage witness; an unreachable witness flags an over-constrained proof or a misclassified signal.
- **Assertion reachability** — every assertion carries antecedent-reachability and full-sequence covers, so a vacuous pass is caught rather than reported as success.
- **Traceability headers** — every assumption carries a structured header recording intent, type, justification, and risk, so a human can audit why each constraint exists and trace it back to the source.
- **Self-identifying notation** — a consistent prefix scheme distinguishes properties, assertions, assumptions, and covers at a glance, independent of the originating medium.

8. Deterministic Configuration

The final rule that makes Gen-SVA reproducible is explicit configuration. Decomposition strategy, latency bounds, vector widths, coverage requirements, verbosity, and gate behavior are fixed up front so generation is deterministic rather than left to the discretion of whatever model or parser produced the output. Two consequences follow: the same requirement yields equivalent assertions across runs and across engines — correctness becomes a property of the rule corpus, not the generator — and the configuration itself becomes documentation of the assumptions the suite was built under, which is essential for sign-off.

9. Worked Illustration: One Requirement, Three Sources

Consider a single behavioral requirement — "every request must eventually be acknowledged" — arriving from three different media. The Gen-SVA claim is that all three converge on the same formal-ready property after the rules are applied.

- **From English**: states intent but no bound. The ambiguity gate forces a latency contract (MAX); "eventually" triggers liveness classification.
- **From a SystemC model**: a transaction-level **`req() / ack()`** pair carries ordering but no clock semantics, which must be lowered to a clocked relationship with confirmed direction.
- **From a waveform image**: a trace showing `ack` three cycles after `req` must be generalized into a bounded window, not hard-coded as "exactly three."

After the rules are applied, all three yield the same property and the same companion set: a bounded liveness assertion, an activation-fairness assumption preventing the request from being starved, an anti-livelock assumption letting the transaction complete, and the coverage witnesses that prove none of it is vacuous:

```
// Convergent formal-ready output (any source)
property p_req_eventually_ack;
  $rose(req) |-> ##[1:MAX] ack;          // bounded liveness
endproperty
ap_req_eventually_ack: assert property (
  @(posedge clk) disable iff (!rst_n) p_req_eventually_ack);

mp_req_fair:        assume property (s_eventually req);
mp_req_no_livelock: assume property (s_eventually always !req);
cp_req_ack_trig:    cover  property ($rose(req));
cp_req_ack_full:    cover  property ($rose(req) ##[1:MAX] ack);
```

The three front-ends could not be more different; the output is identical. The medium chose the parser, but the rules chose the assertion. And had all three been supplied **together**, the result would be stronger still: the waveform pins `MAX` to a concrete value, the English fixes the intent, and the SystemC ordering confirms direction — each closing a gap the others leave open, with any disagreement raised as a clarification rather than silently resolved.

10. Summary of Gen-SVA Rules

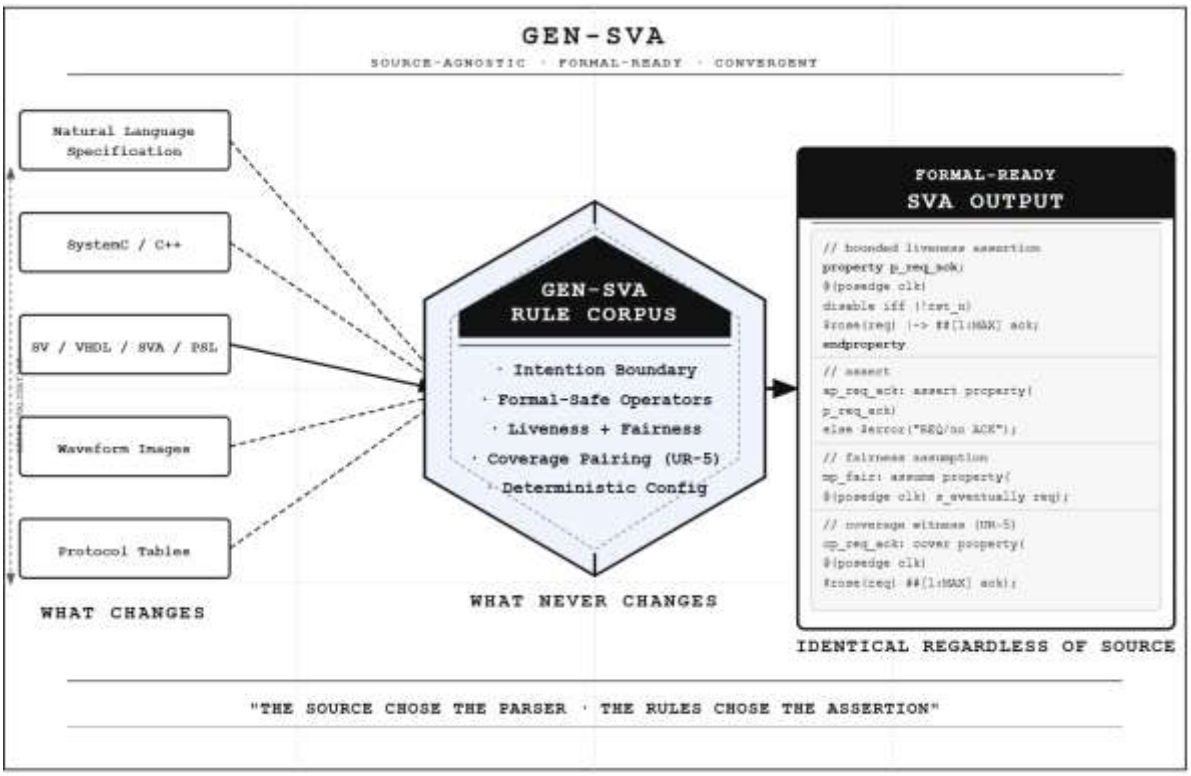
The following rules are mandatory and source-independent. A generator that omits any of them produces simulation-grade, not formal-grade, output.

Rule	What it guarantees
Intention boundary	Inputs become assume; design outputs become obligation. Prevents false-PROVEN over-constraint.
Multi-format corroboration	When several representations of a requirement exist, cross-reference them: agreement raises confidence, disagreement forces clarification.
Formal-safe operators	Bounded windows only; bans `|=>`, until, within, firstmatch, and unbounded delays.
Liveness conversion	"Eventually" lowered to bounded form; pure liveness uses strong().
Fairness & anti-livelock	Companion assumptions stop blocked or toggling inputs from gaming the proof.
Coverage pairing	Every assumption and assertion paired with reachability witnesses; catches vacuity.
Traceability headers	Every assumption documents intent, type, justification, and risk for human audit.
Deterministic config	Bounds, decomposition, and gates fixed up front; output reproducible across engines.

11. Conclusion: The Style Is the Standard

As assertion generation expands beyond English to SystemC, RTL, images, and structured data, the temptation is to build a new generator for each source. The Gen-SVA position is that this is the wrong axis of investment. The source-specific front-end is replaceable; the formal-verification rule set is not. Moreover, where several representations of a requirement coexist, the strongest flow does not pick one — it uses them together, treating their agreement as corroboration and their disagreement as a bug caught early. A generator earns the label "formal-ready" only if, whatever it ingests, it enforces the intention boundary, restricts itself to a formal-safe operator subset, converts liveness into provable form with fairness and anti-livelock support, pairs every assumption and assertion with coverage, and runs under explicit, deterministic configuration.

Stated plainly: the value is not in reading the source — it is in the rules and styles applied after reading it, and in using every available representation of a requirement rather than the first one that suffices. A generator that honors those rules produces assertions that are reproducible, auditable, and tool-portable, and that actually verify what the engineer intended. One caveat remains permanent: these rules guarantee structural soundness, not specification correctness. A latency bound is only as tight as the contract it encodes, and that engineering judgment remains the human's responsibility — the framework makes the judgment explicit and traceable; it does not replace it.



[i] [i] **Knowledge-Driven AI for SystemVerilog Assertions:**
A Verification Framework of Methodologies, Processes, and Coding Rules 2026
Book ships with downloadable RAG-SVA Files
https://SystemVerilog.us

[ii] **DOC9** — the mandatory header block (ASSUMPTION/TYPE/JUSTIFICATION/RISK LEVEL) required on every assume property when FV Mode=YES. Legacy code predates this rule, so its assumptions typically have no such documentation.

INTENTION — the assume-vs-obligation classification discipline (environment signals → antecedent only, DUT signals → consequent only). Legacy properties were often written without this boundary check, so signals may be misplaced across antecedent/consequent without ever having been validated against it.