Do You Need to Know How to Write SVA, or Just Read It?

Why the basics still matter when LLM + RAG writes the assertions, and the expanded requirements-to-SVA process

Ben Cohen · Kais Chibani · systemverilog.us · Ben_Cohen@SystemVerilog.us · November 2026

Abstract

Now that an LLM guided by a rule corpus (LLM + RAG, such as the Cohen-Chibani RAG-SVA ruleset) can generate SystemVerilog Assertions in the best format, with documentation, for both formal verification and simulation, a fair question is whether engineers still need to learn SVA at all. Our answer: you no longer need authoring mastery, but you absolutely need the basics, enough to *read* SVA fluently, as taught by exposure in *Fast-Tracking SVA through Exposure* (Cohen, 2025). The reason is that the process has not removed the engineer; it has moved the engineer from writing to deciding. This paper expands that process, requirements to clarification to SVA, with its two feedback loops, and shows at which gates reading fluency is exercised.

1. The question

Do you need to know how to write SVA, or just read it? For twenty years the answer was "write": assertions existed only if an engineer typed them, so every team needed at least one person with full command of the language, and many teams, lacking one, wrote few assertions or none. The arrival of LLM + RAG generation changes the economics. When the tool writes the properties, what skill must the engineer still hold?

2. The answer: the basics, for reading

We say: you need to know basic SVA, the level *Fast-Tracking SVA through Exposure* teaches by worked examples rather than by language-lawyer study. That means reading, at sight:

- implication (`|->`, and `|-> ##1` in place of `|=>`) and fixed delays (`##N`, bounded ranges `##[m:n]`);
- repetition (`[*n]`, `[*m:n]`) and simple sequence composition;
- `disable iff`, clocking, and what the sampled value of a signal means;
- the difference between an assumption (`assume`, environment), an assertion (`assert`, design obligation) and a cover (reachability);
- strong versus weak properties, and why "still pending at the end" passes for one and fails for the other;
- labels and naming (`ap_`, `mp_`, `cp_`) and what a property's documentation header states.

What you no longer need, unless you choose to specialize, is authoring mastery: sequence algebra corner cases, `first_match` subtleties, local-variable flows, recursive properties, engine internals and vendor quirks. That knowledge took years, and its scarcity was exactly why assertions were under-used. Reading fluency takes weeks.

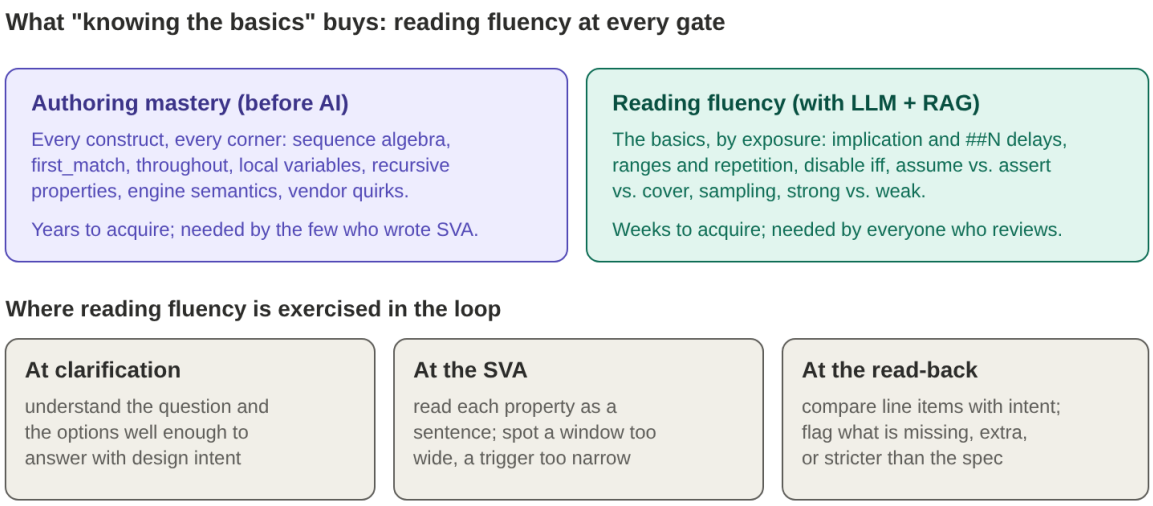


Figure 1. Authoring mastery versus reading fluency, and where reading is exercised in the loop.

3. Why reading is enough: the AI writes in the best format

The claim "just read it" only works if what is generated is worth reading, and this is where the rules matter more than the model. An unguided LLM produces plausible SVA in whatever style it absorbed. Under RAG-SVA, the generated code is constrained to the most readable subset of the language:

- a restricted vocabulary (no multiple implications, no `until`, `within` or `first_match`, no open-ended `##[0:$]`; `throughout` only for fixed-sequence stability), so each property reads as one requirement;
- `|->` `##1` rather than `|=>`, and `!`, &&, `||` in Boolean conditions (OPS1);
- every signal classified as assumed or asserted (the PORT-INTENT table), every assumption carrying a DOC9 header that says why it exists;
- covers paired with every assumption and assertion, so a vacuous pass is visible;
- one source that serves both formal verification and simulation, written for formal even when first run in simulation;
- compilation and simulation with real tools, reported honestly.

Generated this way, the SVA is often *easier* to read than hand-written legacy assertions, because style never varies and intent is stated next to the code. The reading bar the engineer must clear is set by the generator's discipline, and the rules set it low.

4. The expanded process

The sketch behind this paper is a three-step flow, requirements to clarification to SVA, with two arrows looping back. Expanded, the process looks like this:

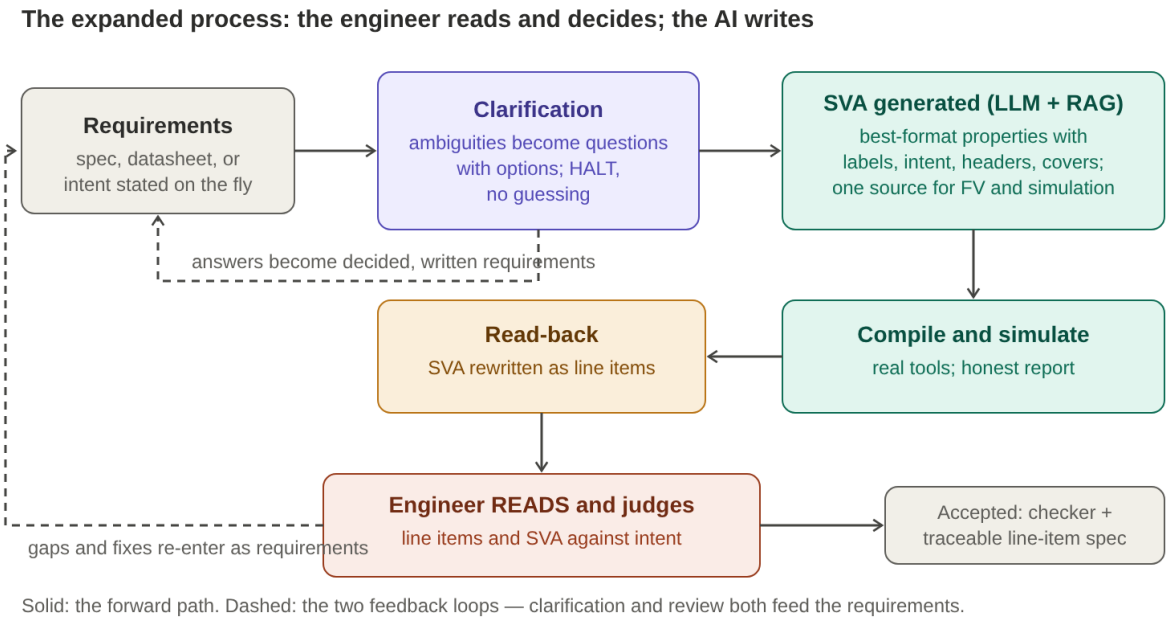


Figure 2. The expanded process. The engineer reads and decides at three gates; the AI writes everything.

1. **Requirements** enter in whatever form exists: a specification, a vendor datasheet, or intent stated on the fly while coding. No form is privileged; the process accepts all three (see *Requirements Before, During, or On the Fly?*).
2. **Clarification.** Before any code, every ambiguity becomes a question with options, and the AI halts rather than guesses (RQM0). **First feedback loop:** each answer is written back into the requirements, so the requirements grow more decided as the process runs. An on-the-fly intent emerges from this gate as a written, decided requirement.
3. **SVA generation.** The LLM, constrained by the rule corpus, writes the properties in the format of Section 3, with their documentation, for FV and simulation at once.
4. **Compile and simulate.** Real tools run the generated files; the report separates what was compiled, what was simulated, what an SVA engine evaluated, and which covers were hit.
5. **Read-back.** The SVA is rewritten as a line-item specification, each item paired with its property.
6. **The engineer reads and judges.** The line items and the SVA are compared with the intent. **Second feedback loop:** everything found, a missing case, a window too wide, an assumption too strong, re-enters at the top as a requirement, and the loop runs again. What survives review is accepted: a checker plus a traceable line-item specification.

Where the reading happens

Note where the engineer's skill is spent in Figure 2: at the clarification gate (understanding the question and answering with design intent), at the generated SVA (reading each property as a sentence), and at the read-back (comparing line items with intent). All three are *reading* acts. None requires the engineer to compose a property from a blank page; all three require the engineer to recognize when a property does not say what the requirement means. That is exactly the skill the basics give.

5. An honest boundary

Reading fluency does not make the engineer dispensable, and the basics are a floor, not a ceiling. Someone on the team should still grow deeper: to write the occasional auxiliary logic the generator asks about, to judge a subtle counterexample, and to extend the rule corpus itself when a review finds a pattern the rules missed (that is how the TOOL EXECUTION and OPS1 rules were added). The experience economy of *AI Efficiency and Engineering Experience* applies: experience moves from typing properties to judging them and to writing the rules.

6. Conclusion

Do you need to know how to write SVA, or just read it? Learn the basics, as in *Fast-Tracking SVA through Exposure*: enough to read any property the generator produces and say whether it matches your intent. Let LLM + RAG write, in the best format, with documentation, for FV and simulation together. The expanded process makes the division of labor explicit: the AI writes at every step, and the engineer decides at three gates, clarification, the SVA, and the read-back, with both feedback loops landing back in the requirements. Reading, not writing, is where the engineering judgment now lives.

References

- B. Cohen, *Fast-Tracking SVA through Exposure: Core Usage, Concepts, AI Integration*, Cohen Publishing, 2025, systemverilog.us.
- B. Cohen, K. Chibani, *Knowledge-Driven AI for SystemVerilog Assertions* (RAG-SVA), Cohen Publishing, 2026, systemverilog.us.
- B. Cohen, K. Chibani, *Requirements Before, During, or On the Fly?*, systemverilog.us/Requirements_Before_During_On_the_Fly_RAG-SVA.pdf.
- B. Cohen, K. Chibani, *AI Efficiency and Engineering Experience*, systemverilog.us/AI_Efficiency_and_Engineering_Experience.pdf.
- B. Cohen, K. Chibani, *Can You Trust AI-Generated Assertions?*, systemverilog.us/Trusting_AI_Generated_SVA_CY62256N.pdf.
- Related papers: systemverilog.us/#papers.

RAG-SVA: personal/academic use per book purchase. Commercial use requires a Commercial License from Cohen Publishing (systemverilog.us).