AI Efficiency and Engineering Experience

Measured AI, the human in the loop, and skills as captured knowledge in chip design and verification

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

Abstract

A question often asked on LinkedIn: how should engineers balance the efficiency AI brings with the value of traditional engineering experience? Our answer has four parts. First, *measured* AI, meaning AI guided by explicit rules and constraints, gives a large gain in efficiency, which lets engineers focus on requirements and verification rather than on writing code. Second, the engineer must stay in the loop to answer the AI's questions, verify its output and refine its rules. Third, experience is not passé; its role changes from typing code to judging code and writing rules. Fourth, a Claude *skill* is the mechanism that turns that experience into knowledge that is captured once, reused on every project, and expanded as lessons are learned. We illustrate each point with the Cohen-Chibani RAG-SVA ruleset and the CY62256N SRAM case study.

1. The question

With the rapid advance of AI, will it mainly make engineers more efficient, or will it fundamentally change traditional design and verification methods? And if it changes them, what happens to the value of experience that took decades to build? The answer we propose is that AI changes *where* experience is applied, and that the efficiency gain is only safe when experience is applied there.

2. What AI does well

Used with the right rules, AI brings two major benefits to chip design and verification.

It finds and clarifies problems in the requirements. It draws on a broad base of public knowledge (standards, datasheets, common practice) and it can state an ambiguity, gap or conflict clearly, as a question with options that the engineer can answer. A requirement such as "data is valid t_AA after the address" can be read in more than one way; a well-guided AI asks which reading is intended instead of choosing one silently.

It codes well and restates what it understood. It writes assertions, RTL models and testbenches; it documents the code; and it can restate its understanding of the code and of the requirements in other forms: line-item specifications, tables, diagrams. These restatements are what make a real human review possible, because a reviewer can compare the restatement against the intent without reading every line of code.

3. Measured AI: rules and constraints

Without rules, an AI fills gaps with plausible guesses, and its guesses can look convincing. In verification that is dangerous: an assertion that compiles, simulates and passes can still check the wrong thing. We

use the term *measured AI* for AI that works inside an explicit rule set: what it may and may not generate, when it must stop and ask, and how it must report what it did.

In the RAG-SVA ruleset these rules include:

- **Halt on ambiguity.** No code is generated while a requirement can be read more than one way; the AI asks, with options.
- **Stated intent.** Every signal is classified as environment (assumed) or design (asserted), and every assumption carries a header saying why it exists.
- **Restricted vocabulary.** Constructs that are easy to misread are banned or restricted, so each property can be read back as one requirement.
- **Covers.** Every assumption and assertion has covers, so a check that never fires is visible.
- **Real tools.** Generated files are compiled and simulated with real tools, and the report separates what was compiled, simulated, evaluated by an SVA engine, and covered.
- **Read-back.** The checker is translated back into a line-item specification for the engineer to review.

4. The efficiency gain, stated honestly

In the CY62256N SRAM case study, Claude with the RAG-SVA rules went from the vendor datasheet to an SVA checker, an RTL model and a testbench in 36 minutes of wall-clock time; the checker was ready 14 minutes after the clarification questions were answered. A read-back specification took under 8 minutes, and a second checker generated only from that read-back took under 10 minutes.

Comparing this with manual effort requires an estimate of how long an engineer would take for the same work, so we state the gain as large (an order of magnitude or more for this kind of task) rather than as a precise multiplier. One case study does not make a general productivity figure. What it does show is where the saved time should go: into getting the requirements right and into verifying the design, the two activities that decide whether silicon works.

5. The human in the loop

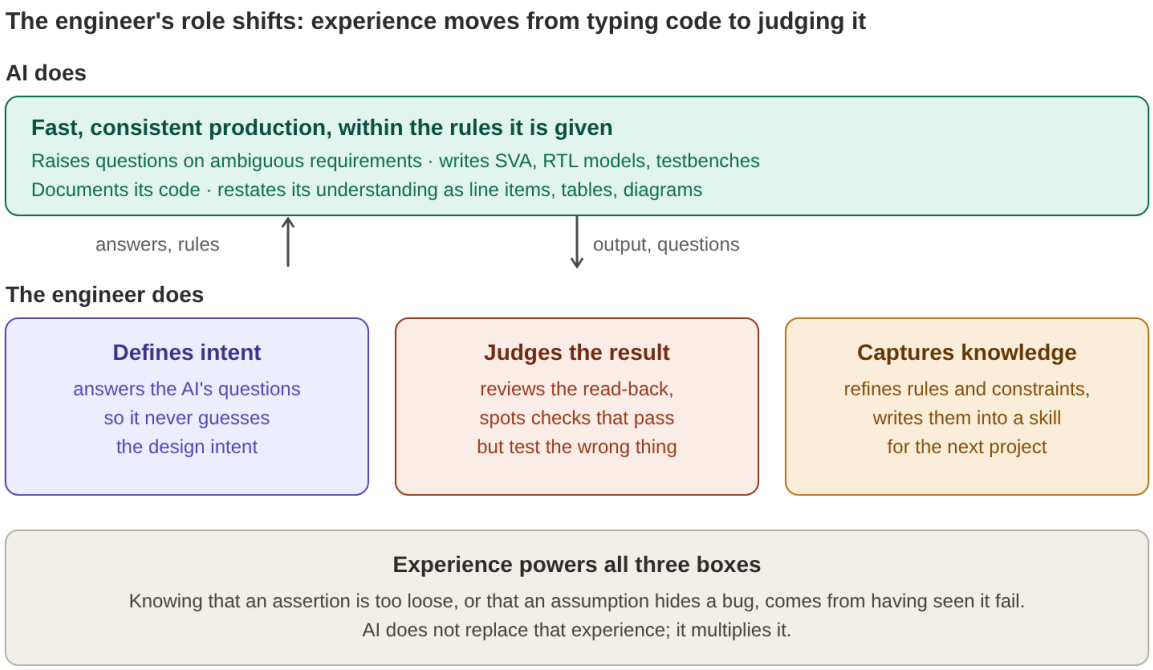


Figure 1. AI produces within its rules; the engineer defines intent, judges the result and captures knowledge.

The engineer stays in the loop for three reasons:

1. **To answer the AI's questions about the requirements**, so it never guesses the design intent. The answers are decisions only the design owner can make.
2. **To verify what the AI produces.** The engineer reviews the read-back against the intent and looks for checks that pass but test the wrong thing, assumptions that are too strong, and requirements that were missed.
3. **To refine the rules and constraints.** When a review finds a problem the rules did not prevent, the fix is not only to correct this output but to add or sharpen a rule, so the problem does not recur.

6. Experience is not passé; its role changes

It is tempting to conclude that if AI writes the code, traditional engineering experience matters less. The opposite is true. Each of the three human roles above depends on experience:

- Knowing which questions about a requirement matter, and answering them correctly, comes from having designed and debugged similar interfaces.
- Recognizing that an assertion is too loose, or that an assumption hides a real bug, comes from having seen those failures before.
- Writing a good rule requires knowing which mistakes are common, which are costly, and which constructs tools and people misread.

What becomes less valuable is spending that experience on typing code by hand. Experience moves from production to judgment and to the rules themselves. AI does not replace experienced engineers; it

multiplies them. An inexperienced engineer with an unconstrained AI produces plausible output faster; an experienced engineer with a well-ruled AI produces correct output faster.

7. Skills: captured knowledge, reusable and expandable

Rules are only useful if they are applied every time, by every engineer, on every project. In Claude this is the role of a **skill**: a folder containing instructions (a `SKILL.md` file) and reference files, which Claude loads automatically whenever a task matches the skill's description. A skill is, in effect, an engineer's or a team's experience written down in a form an AI applies consistently.

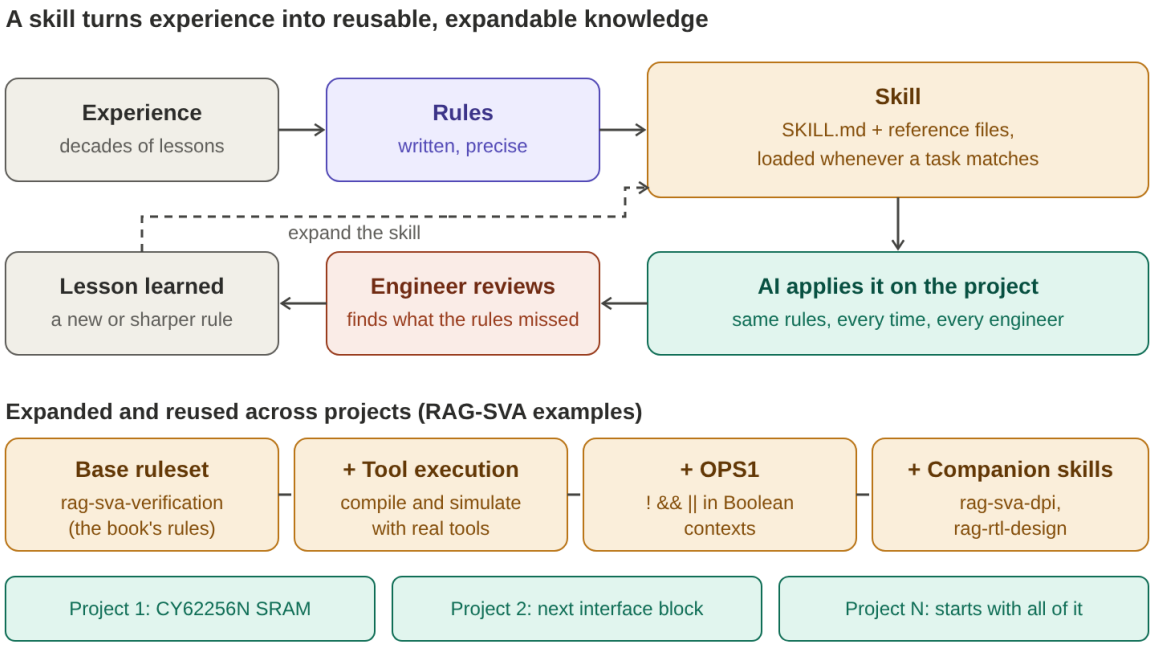


Figure 2. Experience becomes rules, rules become a skill, and each project's lessons expand the skill for the next one.

Captured

The RAG-SVA rules, developed for the book *Knowledge-Driven AI for SystemVerilog Assertions*, are packaged as the `rag-sva-verification` skill: settings, operator rules, intent classification, the clarification gate, decomposition strategy, naming, and a record of past mistakes and why each rule exists. Knowledge that used to live in one person's head, or in a book on a shelf, is now applied whenever an SVA task starts.

Reusable

The same skill serves every project and every engineer who has it. The CY62256N checker was produced under it; the next interface block starts with the same rules, without anyone having to remember or re-explain them. Skills also compose: one task can load several, so a verification skill and a design skill can work together on the same block.

Expandable

A skill is not frozen. When a review finds something the rules missed, a rule is added, and every later task benefits. The RAG-SVA skill grew this way during recent work:

Addition	Why it was added
TOOL EXECUTION rule	Generated code must be compiled and simulated with real tools, and reports must say exactly what was and was not evaluated by an SVA engine.
OPS1 logical-operator rule	Boolean contexts use !, && and ||; bitwise operators are reserved for multi-bit data. Prompted by a review of generated flip-flop code.
rag-sva-dpi (companion skill)	Rules for SystemVerilog/C boundaries: an intent gate for each DPI import, guards against X values and size errors, and a list of gaps that SVA cannot cover.
rag-rtl-design (companion skill)	Draft rules for generating and reviewing synthesizable RTL, designed to work alongside the verification skill.

Each addition came from a concrete review finding: a human judgment, written down once, and then applied automatically. This is the loop in Figure 2, and it is how experience compounds instead of retiring with the engineer.

Governance

Because a skill is a set of text files, it can be reviewed, versioned and licensed like any other engineering asset. Teams should treat changes to a skill the way they treat changes to a coding standard or a verification plan: proposed, reviewed by an experienced engineer, and recorded with the reason for the change.

8. A recommended practice

1. Use AI with rules, never without. Start from an established rule set and package it as a skill.
2. Make the AI ask, not guess. Treat its clarification questions as a review of the requirements.
3. Require a read-back and review it against the intent. Do not accept output that only compiles and passes.
4. Run real tools and keep the report honest about what was actually checked.
5. Turn every review finding into a rule, add it to the skill, and let the next project start from the expanded skill.
6. Put experienced engineers on judgment and rule writing, and pair them with less experienced engineers so the experience is transferred, not only encoded.

9. Conclusion

AI will change traditional verification methods, and it already makes engineers much more efficient. But the efficiency is safe only when AI is measured by rules, when an engineer answers its questions and judges its output, and when the lessons from that judgment are written back into the rules. Experience is what makes all three possible. Skills are how that experience is captured once, reused on every

project, and expanded over time. The balance between AI efficiency and engineering experience is therefore not a trade-off: the efficiency depends on the experience.

References

- B. Cohen, K. Chibani, *Knowledge-Driven AI for SystemVerilog Assertions* (RAG-SVA), Cohen Publishing, 2026, systemverilog.us.
- B. Cohen, K. Chibani, *Can You Trust AI-Generated Assertions? Catching SVA That Passes but Does Not Match the Spec*, systemverilog.us/Trusting_AI_Generated_SVA_CY62256N.pdf.
- B. Cohen, K. Chibani, *From Datasheet to Traceable Requirements: The CY62256N SRAM Example*, systemverilog.us/V5_Vendor_Datasheet_to_Traceable_Requirements_The_CY62256N_SRAM.pdf.
- B. Cohen, K. Chibani, *Requirements Before, During, or On the Fly?*, https://lnkd.in/p/giAMSQUC.
- Anthropic, Agent Skills documentation, docs.claude.com.
- Related papers: systemverilog.us/#papers.

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