Mapping the RAG-SVA Body of Work

A Classification of Ben Cohen and Kais Chibani's Papers on AI-Assisted SystemVerilog Assertion Generation, with Contributions and Links

---

Ben Cohen & Dr. Kais Chibani

VhdlCohen Publishing

systemverilog.us

September 2026

Abstract

Between 2026's book *Knowledge-Driven AI for SystemVerilog Assertions* and the present, close to thirty papers, notes, and case studies applying AI with Retrieval-Augmented Generation for SystemVerilog Assertions (RAG-SVA) have accumulated on systemverilog.us. Individually each is a short, focused piece. Together they no longer present a single, visible shape: a reader arriving at the papers list sees titles but not the argument each one is making, which of them repeat ground already covered, and which contain the load-bearing ideas. This paper classifies that body of work into six clusters — the framework itself, its methodology pillars, worked case studies, legacy modernization, competitive positioning, and the trustworthiness of AI as a collaborator — and states, for each paper, the one contribution that justifies its existence. A cross-cutting matrix then shows which papers exercise which of the framework's core mechanisms (the RQM0 ambiguity gate, the assume/obligation intention boundary, bounded liveness, fairness and anti-livelock constraints, and mandatory coverage), so that a specific mechanism can be traced back to the paper that treats it in depth rather than the paper that merely mentions it. Two link-hygiene issues found while assembling this survey are also reported, since they affect whether a reader can actually reach two of the cited papers from the site's own listing.

This paper deliberately does not repeat the technical content of the papers it classifies; each is one to twenty-six pages, and the point here is the map, not a duplicate of the territory.

Contents

1. Why a map is needed

A single paper is judged on its own terms: does it make its point, is the example clear, is the code correct. A *body* of papers is judged differently — by what it covers as a whole, where it repeats itself, and where it has gaps. Once a papers list passes a certain size, that second judgment becomes hard to make by scanning titles, because a title optimized to be catchy on its own ("It's the Requirements, STUPID!") does not by itself say how that paper's contribution differs from a neighboring one with a similarly styled title ("It's the Methodology, Stupid!", "It's the Source, Stupid!"). The three are, in fact, three different arguments — about requirements, about writing style, and about where a specification comes from — but a reader has no way to know that without opening all three.

This paper is that opening. Section 2 sets out the classification scheme and how papers were assigned to clusters. Sections 3 through 8 are the six clusters, each as a table of title, one-line contribution, and link, followed by a short prose paragraph on what stands out in that cluster. Section 9 is a cross-cutting matrix: for a reader who wants "the paper that treats fairness and anti-livelock constraints in the most depth" rather than "papers in cluster 2," the matrix answers that directly. Section 10 places the whole body against the book that underlies it. Section 11 suggests where the coverage is thin. The full reference list, with links, is at the end.

1.1 Scope and method

Every paper classified here was fetched from systemverilog.us and read (title, abstract or opening section, and — for the shorter pieces — the full text) rather than classified from its title alone; the exceptions are the OVL arbiter, parking-lot, AHB/PSL, and VHDL-FIFO packages, which are distributed as archives of code and Office documents rather than a single formatted PDF, and were classified from their contained readme-equivalent document plus a listing of the archive's files. Three older, purely syntax-level SVA papers linked from the same site — on assertion degeneracy, the `within` operator, and `intersect` versus `throughout`/`until`/`within` — predate the RAG-SVA framework and are not about AI-assisted generation; they are cited only as background reading where a classified paper references them, not classified themselves. A LinkedIn-style post ("Path for engr for effective solutions") is likewise not a paper and is left out.

2. The classification scheme

Six clusters, in the order a reader would logically want them:

1. **Framework** — papers that define RAG-SVA itself, or generalize it beyond a single input format.
2. **Methodology pillars** — papers that go deep on one mechanism or convention inside the framework: an operator, a naming scheme, testbench style, abstraction, determinism, change-impact scope, or solver cost.
3. **Case studies** — papers that apply the framework to one concrete design end to end, usually including a transcript of an actual AI session.
4. **Legacy modernization** — papers specifically about bringing old PSL or VHDL verification assets into the framework.
5. **Positioning** — the one paper that places RAG-SVA against other published LLM-for-SVA frameworks.
6. **AI trustworthiness** — papers about the AI collaborator itself: its failure modes, its legal footing, its tool-independence, and an analogy to a similar AI application outside chip design.

A paper is placed in the cluster whose question it is *primarily* answering. Several papers touch more than one cluster's concerns — the async FIFO case study, for instance, is also a legacy-adjacent argument about specification quality — and where that happens, Section 9's matrix carries the secondary relationship; the cluster tables carry only the primary one.

3. Cluster 1 — Framework

These four papers, read in order, are the fastest path to understanding what RAG-SVA is and why it exists.

Title	Contribution	Link
The RAG-SVA Paradigm Shift in Formal Verification	The framework's own account of itself: the RQM0 ambiguity gate, the PORT-INTENT boundary, bounded-versus-unbounded liveness (Form A / Form B), the FV8/FV9 fairness and anti-livelock infrastructure, and UR-5 coverage pairing, illustrated with one full worked session transcript from a bare requirement to a signed-off checker file.	Paper_June2426.pdf
Gen-SVA: Source-Agnostic Generation of Formal-Ready SystemVerilog Assertions	States and defends the claim that the *input medium* (English, C/SystemC, RTL, a waveform image, a legacy PSL suite, a protocol table) is not what determines assertion quality — the rule set applied at generation time is — and that combining more than one available medium catches errors no single medium would surface alone.	Gen-SVA_Paper_Cohen_KAIS.docx
And Now Is the Time: How AI with RAG-SVA Transforms Requirements, Assertions, Formal Verification, and Testbench Generation	Places AI-with-a-rule-corpus in the personal, historical sequence of tools a working designer has adopted (Karnaugh maps, HDL simulators, synthesis, PSL, SVA, UVM), arguing the adoption pattern is the same, not a novelty.	And_Now_Is_the_Time.pdf
RAG-SVA in the Landscape of LLM-Based Assertion Generation	The framework's self-comparison against six external published or preprint frameworks (see Cluster 5 below; listed here as it is also a framework-definition piece by contrast).	V4_RAG-SVA_Landscape_Comparison.pdf

The RAG-SVA Paradigm Shift paper is the one to hand someone who has read nothing else: it is the only paper that shows the full seven-phase gated workflow, from a two-sentence requirement to a complete, documented checker file, in one continuous transcript. Gen-SVA is its natural second read, because it answers the question the first paper leaves open — "does this only work for English requirements?" — with no.

4. Cluster 2 — Methodology pillars

These papers each isolate one mechanism, convention, or failure mode and treat it in more depth than the framework overview has room for.

Title	Contribution	Link
It's the Requirements, STUPID!	The requirements-first thesis: a flawed requirement compounds through every downstream artifact, and forcing a requirement through multiple representations (English, formal logic, waveform, schematic) surfaces ambiguities no one representation reveals alone.	ItsTheRequirementsStupid.pdf
It's the Methodology, Stupid! Meaningful SVA: Write for Formal, Run Anywhere	The "write every assertion as if formal will read it, even in simulation" thesis, with the specific keyword-to-operator mapping errors (hold versus window, `until`, `within`, and `|=>` versus their bounded, explicit replacements) that the framework's checks catch.	Its_the_Methodology_Stupid.pdf
It's the Source, Stupid! Four Entry Points, One Gated Engine	Names the four realistic starting points for a verification task — clarify an ambiguous spec, generate from scratch, upgrade an existing SVA/PSL suite, reconstruct from RTL and old sim data with no written spec — and argues one gated engine, not four separate tools, must sit behind all of them.	Its_the_Source_Stupid.pdf
Where Do FV Abstraction Techniques Live in a Rule-Governed SVA Methodology?	Draws the boundary between what belongs to the formal tool (cut points, black-boxing, symbolic variables, scoreboards) and what belongs to the rule corpus (the new assumptions and INTENTION flips those abstractions create), with a worked black-boxing example.	fv_abstraction_paper.pdf

Title	Contribution	Link
Determinism in SystemVerilog Assertions for Formal Verification	Defines determinism precisely for a property (one legal pass/fail outcome per state and input set), shows why an `or`-joined antecedent or consequent is the actual problem — not timing or repetition ranges, which are deterministic in SVA's own scheduling semantics — and connects this to the same-session bans on `first_match` and unbounded windows.	Determinism_SVA_FV_Paper.pdf
How to Use the "not" Operator for Formal Verification	A single-operator case study: `not()` is a property operator, legal only in a consequent, and wrapping a whole implication in it silently converts the check into propositional logic — three translations of one English requirement (broken, legal-but-opaque, correct) show the trap and the halt that catches it.	Not_Operator_for_FV_July26.pdf
Naming Conventions for SystemVerilog Design and SystemVerilog Assertions	Compares the framework's own `p_`/`ap_`/`mp_`/`cp_` convention against three other public schemes (an earlier Cohen convention, lowRISC's "Comportable" style used by OpenTitan, and Doulos's "Easier UVM" guidelines) on one shared req/ack example.	Naming_Conventions_Cohen_Chibani_RAG-SVA.pdf
Testbench Styles Supported by RAG-SVA: Directed, Fast Random, and UVM	Argues the three testbench styles are a progression sharing one checker, not competing choices, and states what each style proves and does not prove.	RAG-SVA_Testbench_Styles.pdf
Formal Verification Scope Analysis After a Design Change	A change-impact methodology built from the one framework artifact most useful for it — the PORT-INTENT table — combined with standard commercial FV practice, since the RAG-SVA rule corpus itself has no change-impact section.	FV_Design_Change_Scope_Analysis.pdf
Constraint Overhead in Formal Verification: A Q&A	A solver-complexity Q&A on whether a large set of helper assumptions degrades performance, and whether a combined multi-clause constraint is better than several split ones.	FV_Constraint_Overhead_QA.pdf

The three "It's the ___, Stupid!" papers are easy to mistake for one repeated argument under different titles; they are not. Requirements, Methodology, and Source answer three different questions — what is wrong with the input, what discipline makes the output meaningful, and where realistically does the input come from — and a reader who has only skimmed titles is the one most likely to think one of the three is redundant.

5. Cluster 3 — Case studies

Each of these takes one design, real or drawn from an existing textbook, through the framework end to end, and most include a verbatim transcript.

Title	Contribution	Link
OVL Arbiter: Claude with vs. without SVA-RAG for Formal Verification	A three-way comparison — AI with the RAG-SVA ruleset, the same AI with no rules, and a traditional manually written OVL model — against the Accellera OVL arbiter specification, scoring the RAG output as production-quality and the unguided output as materially weaker on the same input.	ovl_arbiter_ai_rag2.gz
AI-Assisted Formal Verification of a SystemVerilog Parking Lot Controller	Takes a real design from 2009 — assertions the author wrote himself, before the framework existed — and runs it through an audit, repair, and extension pass, producing a compliant checker, a bind statement, a constrained-random testbench, and full documentation; a demonstration that the framework works on the author's own legacy code, not only on fresh requirements.	parking_lot_ai.tar.gz
grnt → req: Claude LLM vs. Claude LLM + RAG	A rule-by-rule violation scorecard between an unguided Claude-generated property and the RAG-compliant equivalent for the same grant/request requirement — clocking placement, the INTENTION boundary, a `$past()` initialization hazard, label prefixes, and a missing fairness constraint are each scored individually.	compare_2files_against_RAG_rules_with_code.pdf

Title	Contribution	Link
Degeneracy, Environment vs. Obligation, and Design Intent: A req/ack Case Study	Reviews a four-line property by hand first — finding temporal degeneracy, an obligation signal in an assumption, and an underspecified hold intent — then reproduces the same review as a verbatim AI session, including one misstep the AI session makes and self-corrects.	req_ack_degeneracy_paper.pdf
Deriving a Formal req/ack Assertion Under the RAG-SVA Methodology	A step-by-step derivation, not just a final answer, for a req/ack pair where `req` must hold until and including the acknowledging cycle and `ack` must arrive two to three cycles later — the paper's value is in the reasoning trace, not the resulting code alone.	req_ack_hold_window_reasoning.pdf
From Ambiguous Requirement to Verified Assertions	A two-lane req/rdy/ack requirement that needed five rounds of clarification — including one genuine gap that survived an initial halt-on-ambiguity pass — distilled afterward into a reusable, fill-in requirement template.	Requirement_Template_Paper.pdf
Async FIFO Decomposition: From RTL and SVA to a Black-Box Specification — and Back	Starts from the async FIFO of an earlier Cohen book chapter, generates SVA and documentation from the original RTL, distills that into a black-box specification classified by verification category (Safety, Functional, Environment Assumption, Obligation/Flag Contract) across all thirteen requirements, and then regenerates a working RTL model and testbench from the specification alone, with no access to the original files — the closed loop is the empirical result.	fifo_async_ragsva_paper.pdf

Title	Contribution	Link
Same Design, New Node: What Must Change in the SVA and the RTL When a Verified 5 nm Design Moves to 3 nm	A technology-migration case study rather than a requirements case study: which SVA bounds and RTL pipeline stages a process-node change actually moves, why a single scaling factor is wrong, and what an AI assistant can and cannot re-derive without a timing report.	Same_Design_New_Node.pdf

Two pairs in this cluster are easy to conflate and worth distinguishing explicitly. The **degeneracy paper** and the **hold-window paper** both work a req/ack-family requirement, but the first is about catching what is *wrong* with a given property (a review), while the second is about *deriving* a correct one from a stated requirement (a derivation) — review and derivation are different skills, and the framework treats them as different sessions. The **OVL arbiter paper** and the **grnt→req comparison** both pit RAG-guided output against unguided output, but the arbiter paper adds a third arm (manual, human-written code) that neither of the other comparison papers has, which is what makes it the stronger of the two as evidence.

6. Cluster 4 — Legacy modernization

Title	Contribution	Link
Converting Legacy PSL-SIM to SVA-FV-Ready Code	Converts 2004-era PSL assertions for an AHB bus compliance layer and a memory slave controller (from Cohen's 2004 PSL book) into FV-ready SVA, with a table mapping each RAG-SVA rule category to the specific legacy defect it catches: unbounded boundaries, mixed input/output constraints in one expression, weak sequences that pass vacuously under FV, and a total absence of fairness infrastructure or coverage in the original.	PSL2SVA_convert.tar.gz
Converting Legacy VHDL to SVA-FV-Ready Code + SPECS	Takes a VHDL asynchronous FIFO and produces both a formal English specification (detailed enough to regenerate the design, per the paper's own stated goal) and the corresponding SVA, plus a follow-up extension adding a data-integrity requirement and the fairness constraints that requirement implies.	vhdl2sva.tar

Both papers report the same qualitative finding from two different legacy formats: the defect that RAG-SVA catches most reliably in old verification code is not a syntax error but a *soundness* gap invisible to simulation — a weak sequence, a missing fairness constraint, an assumption that was never checked for over-constraint — because simulation-era code was never run against a formal tool that would have exposed it.

7. Cluster 5 — Positioning

Title	Contribution	Link
RAG-SVA in the Landscape of LLM-Based Assertion Generation: Constrain-at-Generation vs. Generate-then-Repair	The one paper that names and cites external, competing frameworks — AssertLLM, NVIDIA's AssertionForge, LISA, a knowledge-graph-based multi-agent approach, and Spec2Assertion, plus a survey covering ChIRAAG, LAAG-RV, AssertGen, SANGAM, DeepAssert, and VERT — and states plainly where RAG-SVA's constrain-at-generation approach differs from their more common generate-then-repair pattern, where it is at a disadvantage (no published benchmark numbers of its own), and what would strengthen it.	V4_RAG-SVA_Landscape_Comparison.pdf

This is the only paper in the set that engages with the external published literature by name rather than by reference to the authors' own prior work, and it is also the only one that names a weakness (the absence of a published quantitative benchmark) rather than only a strength. A reader wanting to place RAG-SVA against the wider 2025–2026 wave of LLM-for-SVA research should start here.

8. Cluster 6 — AI trustworthiness

Papers in this cluster are about the AI collaborator, not about a specific assertion or design.

Title	Contribution	Link
Frame Drift in Multi-Turn LLM Interaction, and Why RAG-SVA Rule Corpora Already Defend Against It	Distinguishes a real risk (frame drift: a sequence of individually reasonable turns incrementally shifts what question is actually being answered) from a folk-model risk that does not describe how conditioned generation works (associative drift, a word pulling in embedding-neighbor concepts), and argues the framework's halt-don't-guess discipline and rule-ID citation already defend against the real risk, though it was not designed with that framing in mind.	Frame_Drift_and_RAG_SVA_Discipline.pdf

Title	Contribution	Link
What if AI Scribe for Doctors Were Used for AI Scribe for Chip Design and Verification?	An extended analogy: the capture-transcribe-extract-structure-integrate-sign-off pipeline built for medical ambient scribes maps almost directly onto the documentation burden in chip design and verification, walked through stage by stage.	AI_scribe_medical_vs_chip_comparison.pdf
NOTES: Is 1800 in LLMs?	A Q&A on two adjacent questions: whether an LLM's fluency with SystemVerilog comes from having absorbed the IEEE 1800 standard's practiced use (yes, largely) rather than its copyrighted text verbatim (no, and should not), and the more unsettled question of whether IEEE/Accellera's licensing terms permit using the standards themselves as retrieval material for RAG.	Q_1800_Rag.pdf
The Value of This Book Without Any Particular AI Tool	Answers a workplace-policy objection directly: the book's judgment (how to classify a signal's intention, how to write a sound assumption, the concept of determinism and of anchoring a property to a clock and reset) is manual-transferable and tool-agnostic, so losing access to a specific AI tool does not erase the book's value.	rag_sva_book_value.pdf

The frame-drift paper is the most technically substantial of the four, because it makes a falsifiable claim about how autoregressive generation actually behaves (conditioned on the full context, not on a leaking word-association chain) and then connects that claim to a specific engineering defense. The other three are shorter and more occasional — a Q&A, an analogy, a rebuttal to a specific objection — but each closes a real question a reader or a skeptical colleague would otherwise be left holding.

9. Cross-cutting matrix: which paper treats which mechanism

The framework has five mechanisms that recur across nearly every case study: the RQM0 ambiguity gate, the assume/obligation INTENTION boundary, bounded-versus-unbounded liveness handling, the FV8/FV9 fairness and anti-livelock constraints, and UR-5 mandatory coverage pairing. Most papers touch several in passing while a case study is unfolding; the table below marks a mechanism only where the paper treats it as a central subject, not a passing mention.

Paper	RQM0 ambiguity gate	INTENTION boundary	Bounded liveness	Fairness / anti-livelock	UR-5 coverage
RAG-SVA Paradigm Shift	Central	Central	Central	Central	Central
Gen-SVA	Central	Central	—	—	Central
It's the Requirements, STUPID!	Central	—	—	—	—
It's the Methodology, Stupid!	Central	Central	—	—	—
It's the Source, Stupid!	Central	Central	—	—	—
FV Abstraction Techniques	—	Central	—	—	—
Determinism in SVA	—	—	—	—	—
The "not" Operator	Central	Central	—	—	—
OVL Arbiter comparison	Central	Central	Central	Central	Central
Parking Lot Controller	Central	Central	—	Central	Central
grnt → req comparison	—	Central	—	Central	—
Degeneracy req/ack case study	Central	Central	—	—	—
Hold-window req/ack derivation	Central	Central	Central	—	—
Ambiguous-to-Verified Template	Central	Central	Central	—	—
Async FIFO Decomposition	Central	Central	—	—	Central
Legacy PSL-SIM Conversion	Central	Central	Central	Central	Central
Legacy VHDL Conversion	Central	Central	—	Central	—
Same Design, New Node	—	—	—	—	—

Two gaps are visible immediately. First, **Determinism in SVA** and **Same Design, New Node** are the only two papers that engage none of the five core mechanisms directly — the first is about or-joined property structure, which is a different axis of correctness, and the second is about physical timing derivation, which the framework's mechanisms do not address at all. Neither gap is a defect; they are simply evidence that the body of work has expanded past the five-mechanism core into adjacent territory (state-space structure, physical implementation). Second, only three papers give the fairness/anti-livelock machinery central treatment outside the framework paper itself — the OVL arbiter comparison, the parking-lot controller, and the two legacy-conversion papers — which is thin coverage for a mechanism the framework paper calls out as one of its more novel contributions (Section 6 of the Paradigm Shift paper). Section 11 returns to this.

10. How the papers relate to the book

Every paper in Clusters 1 through 4 either directly cites, or works from a technique documented in, *Knowledge-Driven AI for SystemVerilog Assertions* (Cohen and Chibani, 2026). The relationship is consistent across the set: the book states a rule once, with its justification; a paper then either (a) explains why the rule matters through a single deep example (the methodology-pillar papers), (b) shows the rule catching a specific real defect in a specific design (the case studies and legacy-modernization papers), or (c) extends the rule's reach to a new input format or a new comparison context (the framework and positioning papers). None of the classified papers restates the book's rules from scratch; each assumes the reader either has the book or is willing to take the cited rule on faith for the paper's own purpose. This is by design — the papers are companions to the book, not summaries of it — but it does mean a reader with no access to the book will occasionally hit a named rule (`RQM0`, `UR-5`, `FV-FA-1`) with no local definition, and will need to either consult the Paradigm Shift paper (Cluster 1), which defines the core rules in one place, or the book itself.

11. Where the coverage is thin

Two observations follow from the classification rather than from any single paper.

Fairness and anti-livelock treatment is concentrated in case studies, not isolated. Unlike determinism, the `not()` operator, and naming conventions, which each have a dedicated methodology-pillar paper, FV8/FV9 fairness and anti-livelock constraints are currently explained only inside larger case studies and the framework overview. A dedicated paper working several fairness/livelock patterns side by side — activation blocking, continuous-toggle livelock, and the pulse-versus-hold distinction the Paradigm Shift paper's worked example already surfaces — would match the depth given to the other mechanisms.

The legacy-modernization cluster has two entries and could bear a third. PSL-to-SVA and VHDL-to-SVA are covered; a symmetric "SVA/PSL back to VHDL/PSL" direction is listed on the site's papers page by title only ("Converting SV/SVA to VHDL/PSL") but, per Section 11 above, does not yet resolve to distinct material. If that direction is a live piece of work rather than a labeling artifact, it would complete the modernization cluster's coverage of both conversion directions.

12. Conclusion

Read end to end rather than as a scrolling list, the RAG-SVA body of work has a clear shape: one paper that defines the framework in full, a wider one that generalizes it beyond English input, ten papers that each deepen one mechanism or convention, eight case studies that apply it to real or textbook designs (including, most recently, a technology-migration case rather than a requirements case), two legacy-modernization papers, one paper that places the whole effort against the external published literature, and four papers about the AI collaborator's own behavior and limits. The classification in Sections 3 through 8, the matrix in Section 9, and the gap analysis in Section 11 together answer the question that prompted this survey: what is actually in each paper, and where does its contribution sit relative to the others.

References

1. B. Cohen and K. Chibani, *Knowledge-Driven AI for SystemVerilog Assertions: A Verification Framework of Methodologies, Processes, and Coding Rules*, VhdlCohen Publishing, 2026. Paper copy | PDF | TOC and sample
2. B. Cohen and K. Chibani, The RAG-SVA Paradigm Shift in Formal Verification, systemverilog.us, 2026.
3. B. Cohen and K. Chibani, Gen-SVA: Source-Agnostic Generation of Formal-Ready SystemVerilog Assertions, systemverilog.us, 2026.
4. B. Cohen and K. Chibani, And Now Is the Time: How AI with RAG-SVA Transforms Requirements, Assertions, Formal Verification, and Testbench Generation, systemverilog.us, 2026.
5. B. Cohen and K. Chibani, RAG-SVA in the Landscape of LLM-Based Assertion Generation: Constrain-at-Generation vs. Generate-then-Repair, systemverilog.us, 2026.
6. B. Cohen, It's the Requirements, STUPID! A Methodology Paper: How AI and RAG Transform Requirements from English to Silicon, systemverilog.us, 2026.
7. B. Cohen and K. Chibani, It's the Methodology, Stupid! Meaningful SVA: Write for Formal, Run Anywhere, systemverilog.us, 2026.
8. B. Cohen and K. Chibani, It's the Source, Stupid! Four Entry Points, One Gated Engine, systemverilog.us, 2026.
9. B. Cohen and K. Chibani, Where Do FV Abstraction Techniques Live in a Rule-Governed SVA Methodology?, systemverilog.us, 2026.
10. B. Cohen and K. Chibani, Determinism in SystemVerilog Assertions for Formal Verification, systemverilog.us, 2026.
11. B. Cohen and K. Chibani, How to Use the "not" Operator for Formal Verification: A Case Study in Ambiguous English, Legal SVA, and the Halt That Catches Both, systemverilog.us, 2026.
12. B. Cohen and K. Chibani, Naming Conventions for SystemVerilog Design and SystemVerilog Assertions: A Comparative Reference for the Cohen-Chibani RAG-SVA Framework, systemverilog.us, 2026.
13. B. Cohen and K. Chibani, Testbench Styles Supported by RAG-SVA: Directed, Fast Random, and UVM, systemverilog.us, 2026.
14. B. Cohen and K. Chibani, Formal Verification Scope Analysis After a Design Change, systemverilog.us, 2026.
15. B. Cohen and K. Chibani, Constraint Overhead in Formal Verification: A Q&A on Helper Assumptions, Constrained Signals, and Combined vs. Split Constraints, systemverilog.us, 2026.
16. B. Cohen and K. Chibani, OVL Arbiter: Evaluation of Claude with and without SVA-RAG for Formal Verification, systemverilog.us, 2026.
17. B. Cohen, AI-Assisted Formal Verification of a SystemVerilog Parking Lot Controller: A Process Paper, systemverilog.us, 2026.
18. B. Cohen, grnt → req: Claude LLM vs. Claude LLM + RAG, systemverilog.us, 2026.
19. B. Cohen, Degeneracy, Environment vs. Obligation, and Design Intent: A req/ack Case Study Comparing Expert Manual Review Against an AI-Assisted RAG-SVA Session, systemverilog.us, 2026.
20. B. Cohen and K. Chibani, Deriving a Formal req/ack Assertion Under the RAG-SVA Methodology: A Worked Example: Hold-Until-Ack and a Bounded Response Window, systemverilog.us, 2026.

21. B. Cohen and K. Chibani, From Ambiguous Requirement to Verified Assertions: A Case Study and Reusable Template for SVA Requirement Writing, systemverilog.us, 2026.
22. B. Cohen and K. Chibani, Async FIFO Decomposition: From RTL and SVA to a Black-Box Specification — and Back, systemverilog.us, 2026.
23. B. Cohen and K. Chibani, Same Design, New Node: What Must Change in the SVA and the RTL When a Verified 5 nm Design Moves to 3 nm — and What an AI Assistant Can and Cannot Re-Derive from Technology Data, systemverilog.us, 2026.
24. B. Cohen and K. Chibani, Converting Legacy PSL-SIM to SVA-FV-Ready Code, systemverilog.us, 2026.
25. B. Cohen and K. Chibani, Converting Legacy VHDL to SVA-FV-Ready Code + SPECS, systemverilog.us, 2026.
26. B. Cohen, Frame Drift in Multi-Turn LLM Interaction, and Why RAG-SVA Rule Corpora Already Defend Against It, systemverilog.us, 2026.
27. B. Cohen, What if AI Scribe for Doctors Were Used for AI Scribe for Chip Design and Verification?, systemverilog.us, 2026.
28. B. Cohen, NOTES: Is 1800 in LLMs?, systemverilog.us, 2026.
29. B. Cohen and K. Chibani, The Value of This Book Without Any Particular AI Tool, systemverilog.us, 2026.
30. IEEE Std 1800, *SystemVerilog: Unified Hardware Design, Specification, and Verification Language*.
31. Accellera Systems Initiative, *Open Verification Library (OVL) Language Reference Manual*, v2.

The complete, current papers and books list is maintained at systemverilog.us.