Async FIFO Decomposition: From RTL and SVA to a Black-Box Specification — and Back

A RAG-SVA Case Study in AI-Ready Specification Design

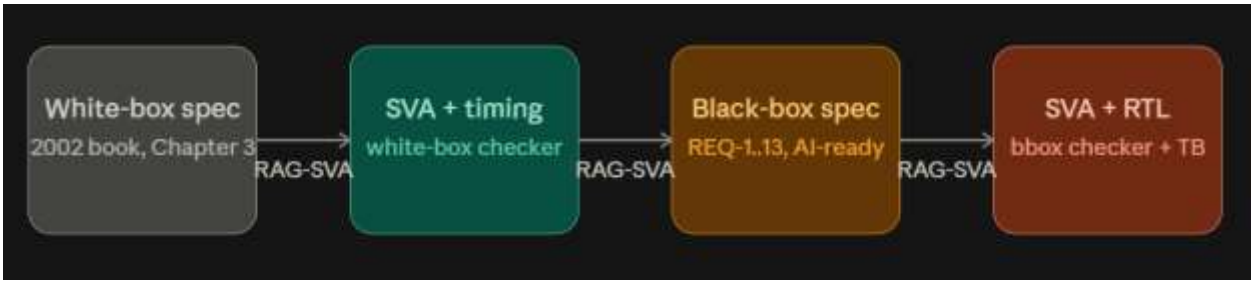
Ben Cohen and Kais Chibani — systemverilog.us

Abstract

This paper documents a small but complete case study in applying RAG-SVA — the retrieval-augmented, rule-governed methodology for AI-assisted SystemVerilog Assertion generation described in *Knowledge-Driven AI for SystemVerilog Assertions (Cohen & Chibani, 2026*) — to a design pulled directly from an earlier work: the asynchronous FIFO from Chapter 3 of *Real Chip Design and Verification Using Verilog and VHDL (Cohen, 2002),* a design originally due to Cliff Cummings and included in that book with his permission. Starting from that chapter, RAG-SVA was used to:

- 1. Generate formal SVA and timing documentation directly from the original RTL.
- 2. Distill the chapter and that SVA into a new, AI-ready black-box specification.
- 3. Regenerate a working RTL model and testbench from that specification alone — with no access to the original source files.

A key contribution of step (2), and the focus of this paper, is that RAG-SVA did not just restructure the chapter's prose into a list — it classified every resulting requirement with a verification-category qualifier (*Safety, Functional, Environment Assumption, Obligation/Flag Contract*) and paired each with a concrete verification strategy, consistently and automatically, across all thirteen requirements. Step (3) is the paper's empirical result: it is a working demonstration that the resulting specification carries enough structured, unambiguous information to reproduce a design independently, which is a stronger property than anything a narrative textbook chapter, however well written, can offer a downstream tool.



1. The Original Specification

Chapter 3 of the 2002 book is, by the standards of its time and purpose, an excellent specification for a human reader. It presents the asynchronous FIFO problem clearly: the difficulty of tracking fullness across two independent clock domains, the hazard of synchronizing a binary counter through a metastable boundary, and the resulting design decision to use Gray-code pointers and a two-flop synchronizer in each direction. The chapter walks through the pointer-equality definition of **full** and **empty**, the resulting "pessimistic" flag behavior, and then presents complete VHDL and Verilog models together with a simple, non-exhaustive testbench and simulation waveforms.

What it does not do — because it was never meant to — is separate the specification from the architecture, or present the requirements in any form other than continuous prose interleaved with implementation code. The **full/empty** contract, the overflow/underflow responsibility split between DUT and environment, and the CDC latency behavior are all stated correctly, but they are stated once, in running text, alongside the RTL that happens to implement them. A human engineer reads this chapter start to finish and comes away with a coherent mental model. A tool — or an AI system — asked to act on this chapter has to first perform its own, unguided extraction of "what is required" from "how this particular implementation happens to achieve it," with no signal table, no requirement identifiers, and no explicit statement of which behaviors are architecture-specific versus which are the actual contract.

2. Three RAG-SVA Passes

2.1 RTL + design → SVA + waveforms

RAG-SVA first read the original chapter directly and produced *fifo_async_fpv_checker*, a white-box formal checker bound to the DUT's internal pointers, memory, and addresses. This pass produced properties tied to the actual gray-code implementation — no-overflow/no-underflow bounds using the CHECK-20-compliant wraparound-subtraction pattern, flag-correctness properties checked against the pointer-equality definition in the text, and a single-bit-change invariant on the Gray-code pointers, which is the specific CDC-safety property the whole design exists to guarantee. Read and write timing diagrams were then generated in two forms — a graphical waveform and a machine-parseable T-Column ASCII form — each tied explicitly back to the properties they illustrate *(for generated timing diagrams see fifo_async_requirements_v1.docx)*.

2.2 Chapter 3 and SVA → black-box specification

The chapter and the derived SVA were then distilled into *fifo_async_requirements_v1.docx*: a 7-column port/parameter signal table, thirteen numbered requirements (REQ-1 through REQ-13), and the same two-form timing diagrams. Everything architecture-specific — *pointer encoding, synchronizer stage count, memory organization* — was deliberately excluded.

This is also the step where RAG-SVA's classification work happens, and it is worth dwelling on. For every requirement it extracted, RAG-SVA assigned two things:

(1) A verification-category qualifier
Safety, Functional, Environment Assumption, or Obligation/Flag Contract

(2) A concrete verification strategy
Formal, Formal + simulation, or Environment assumption, in each case as specific as the requirement warrants.

This is not a cosmetic labeling exercise. Deciding whether a given sentence in a design chapter describes a safety property, a functional behavior, or a constraint on the environment rather than the DUT requires understanding the requirement's role in the design — and doing this consistently across a full requirement set, by hand, is exactly the kind of repetitive, judgment-heavy task where manual work drifts (a tired reviewer classifies REQ-4 as Safety in one pass and Environment Assumption in another) while a rule-governed AI pass does not.

2.2.1 Examples

The following two requirements, reproduced exactly as RAG-SVA produced them, illustrate the classification qualifier and verification strategy together:

REQ-2 *(Safety / Reset)*
Statement: On deassertion of `rrst_n`, the read-side interface shall reflect the empty condition (`rempty` = 1) before any data has been written and become visible.
Verification: Formal (safety), reset-scoped.

REQ-3 *(Functional / Write)*
Statement: While `wen` is asserted (active-low, `wen` = 0) and `wfull` = 0, the value present on `wdata` at that `wclk` edge shall be accepted into the FIFO for later retrieval via `rdata`, preserving order relative to all other accepted writes.
Verification: Formal + simulation.

A third example shows a qualifier the first two do not: an Obligation/Flag Contract, where the DUT owes an observable output behavior to the environment, and where the verification strategy has to account for the design's asynchronous, conservative nature rather than demanding exact-cycle behavior:

REQ-7 *(Obligation / Flag Contract)*
Statement: `wfull` shall assert whenever the FIFO holds `FIFODEPTH` not-yet-read entries, and shall never fail to assert under that condition. Because `wclk` and `rclk` are asynchronous, `wfull` may reflect a conservative (pessimistic) view and may remain asserted for a bounded number of extra write-clock cycles after room becomes available, but shall never under-report a full condition.
Verification: Formal (safety, two-state conservative bound).

Three requirements, three distinct qualifiers, three appropriately different verification strategies — REQ-2 is a one-shot reset check, REQ-3 is a per-transaction functional contract verifiable by more than one method, and REQ-7 is a safety property whose verification strategy has to explicitly accommodate a bounded, non-exact timing allowance.

RAG-SVA produced all thirteen requirements to this same standard in a single pass, each correctly qualified and each paired with a verification strategy suited to what the requirement actually claims — not a generic "verify this" placeholder repeated thirteen times.

2.3 Black-box specification → RTL + TB

Working from only that specification — no access to the original PDF, no internal signal names — RAG-SVA produced a second, independent SVA checker, *fifo_async_bbox_fpv_checker*, which proves the same requirements using a self-contained reference model (*a shadow memory plus two event counters*) built purely from observed port activity. Notably, this black-box checker proves strict FIFO ordering (REQ-11) end to end, a property the original white-box checker never attempted. A companion quick-sanity testbench, *fifo_async_quicktb.sv*, was then generated: a from-scratch reconstruction of the DUT RTL (module *graycntr* and module *fifo_async*), the bind statement attaching the checker, and constrained-random stimulus — built entirely from the specification's REQ statements and signal table.

3. Why the AI-Generated Black-Box Spec Is Better

4. Conclusion

The strongest argument for the new specification's quality is that it was tested the way a specification should be tested: by handing it, alone, to a process that had to act on it without the benefit of the original design. It passed — an independent RTL model, an independent SVA checker proving properties (including data ordering) the first checker never attempted, and a working sanity testbench all came out the other side. The original 2002 chapter remains a well-written and correct account of the design for a human reader; what this exercise adds is a companion artifact built specifically so that an AI system, or any downstream tool, can act on it directly. The same three-pass RAG-SVA workflow — RTL to SVA, SVA and source to black-box spec, spec back to RTL and TB — generalizes as a repeatable method for turning existing, human-oriented design documentation into AI-ready specifications.

Appendix: File Manifest

Available at https://systemverilog.us/Async_fifo/ async_fifo.tar.gz
or individually with https://systemverilog.us/Async_fifo/*File_name*

Date	File	Size	What it is
Originally 2002	`Cohen_async_fifo_book.pdf`	1,821,676 B	Async FIFO chapter from the book. Defines an async FIFO with permission from Cliff Cummings; includes requirements, code, and TB in VHDL and Verilog.
2002	`real_chip_ch3.tar`	—	Book code archive for the async FIFO.
9/1/2026	`fifo_async_fpv_v3.sv`	15,944 B	RAG-SVA read Cohen_async_fifo_book.pdf and produced fifo_async_fpv_checker (white-box).
9/2/2026	`fifo_async_timing_session.pdf`	55,969 B	Creation of timing waveforms from the SVA.
9/2/2026	`fifo_async_session.docx`	—	Session transcript of the above step.
9/2/2026	`fifo_async_requirements_v1.docx`	148,296 B	Book spec + SVA distilled into a new black-box FIFO specification, written to be AI-readable.
9/2/2026	`fifo_async_bbox_fpv.sv`	12,970 B	RAG-SVA took the black-box spec and produced FPV-ready SVA: module fifo_async_bbox_fpv_checker.
9/2/2026	`fifo_async_quicktb.sv`	8,073 B	RAG-SVA creation of a fast TB from the black-box spec: module graycntr, module fifo_async (reconstructed), bind fifo_async fifo_async_fpv_checker, module fifo_async_quicktb.