

From Specification to Proof: AI Agent Automation of AEP and FPV Workflows

Teodora Kostic¹
Advanced Micro Devices, Inc.
Bulevar Mihajla Pupina 115,
Belgrade, Serbia
teodora.kostic@amd.com
phone number: +381606130598

Bojan Dmdarevic²
Advanced Micro Devices, Inc.
Bulevar Mihajla Pupina 115,
Belgrade, Serbia
bojan.dmdarevic@amd.com

Liya Liu
Advanced Micro Devices, Inc.
84 Hines Road, Suite #300
Ottawa, ON K2K 3G3
liya.liu@amd.com

Parmar JayeshKumar Parmar
Advanced Micro Devices, Inc.
Prestige Technostar, Doddanakundi
Industrial Area 2, Brookefield
Bengaluru, Karnataka 560048
parmarjayeshkumar.parmar@amd.com

ABSTRACT

Automatically Extracted Properties (AEP) and Formal Property Verification (FPV) are essential for establishing functional correctness in semiconductor designs, yet their workflows remain labor-intensive and expertise-dependent. Existing AI-assisted methods automate isolated tasks or require repeated prompting. This paper presents an AI-agent framework that automates end-to-end AEP and FPV while retaining engineer review at key checkpoints. From natural-language design specifications, the agent extracts block-level intent, derives candidate AEP assumptions, and creates a traceable property plan. It then generates SystemVerilog Assertions (SVA), constraints, modeling code, tool scripts, and verification-status reports. After formal execution, the agent analyzes failed or inconclusive properties, correlates results with RTL and specification intent, diagnoses likely causes, and proposes corrections through a Model Context Protocol (MCP) server. The framework was applied to the formal verification of four production semiconductor modules using Synopsys VC Formal. Across the evaluated modules, agent-assisted property planning and implementation required an estimated 2 hours. For context, the conventional block-level FPV cycle, including formal execution, failure debug, review, and sign-off, required approximately 7 working days. Following design changes, the estimated property-regeneration effort decreased from 3 working days to 2 hours. The agent also consolidated redundant assumptions, generated cover properties that provide explicit targets for assessing key-scenario coverage, reduced internal modeling variables, and simplified helper logic. The methodology is also applicable in verification using other formal tools, i.e. Cadence JasperGold.

Keywords: formal verification, AI agent, SystemVerilog Assertions, large language models, design verification automation

1 Introduction

In industrial verification flows, formal verification (FV) uses mathematical algorithms to exhaustively determine whether a design satisfies its specification. It complements traditional simulation-based methodologies through diverse

applications. Among them, Automatically Extracted Properties (AEP) [1] can identify conditions such as floating buses, arithmetic overflow, finite-state machine (FSM) livelock, and FSM deadlock early in the design-verification process, before a simulation testbench is available. Formal Property Verification (FPV), by contrast, checks user-defined, rather than tool-extracted, SystemVerilog Assertions (SVA) against the functionality implemented in the target design and is widely used in FV activities.

A major challenge in block- and submodule-level formal verification is the limited availability of detailed design criteria at the beginning of development. Designers typically prioritize implementation and defer detailed documentation because the implementation may change before the block is delivered to verification engineers. Thus, development is often guided by a high-level specification, while the design criteria are refined through iterative discussions between design and verification engineers. This process requires repeated coordination and delays the start of verification.

Although AEP does not require engineers to write assertions, appropriate constraints must be added to prevent spurious failures caused by invalid input combinations. Determining the correct AEP constraints requires designers and verification engineers to align on the intended design behavior. For FPV, the workflow comprises the following four sequential activities:

- (1) planning properties that verify the target functionality,
- (2) implementing the properties in SVA according to a defined rule set,
- (3) executing the formal tool, and
- (4) debugging falsified properties. Each activity requires substantial domain expertise and engineering effort.

Applying either AEP or FPV to a design under test (DUT) is traditionally labor-intensive and requires substantial formal verification expertise. For security-critical blocks involving only a few features, property planning alone can require more than five hours per block, with property implementation, formal execution, and failure debug requiring additional engineering effort.

Recent advances in large language models (LLMs) and agentic AI frameworks provide an opportunity to automate

¹ This author is AMD contractor from Capgemini (<https://www.capgemini.com/ca-en/>)
²

this end-to-end workflow. Prior work has investigated AI assistance for individual formal verification tasks. Sultan et al. [2] introduced an AI- and syntax-rule-based process that generates syntactically correct computation tree logic properties and propagates requirement coverage from formal verification results to SysML requirement diagrams. However, that approach does not interpret natural-language specifications or close the loop after formal-tool execution because automated failure diagnosis and debug remain outside its scope. PropGen [3] generates transparent SVA files for exhaustive FSM verification from Visio state diagrams rather than RTL; however, it targets diagram-based specification input and remains prompt-driven, with human involvement at each step.

The agent presented in this paper extends the earlier work reported in [4] from automated formal-verification test setup and prompt-based SVA generation to a specification-driven framework that supports supervised autonomous execution between engineer-review checkpoints across complete AEP and FPV workflows, including specification generation, property planning, SVA implementation, formal-execution support, failure diagnosis, and debug guidance.

The proposed agent consumes natural-language specifications supplied by designers, together with meeting transcripts when available, to generate a block-level specification, derive candidate AEP assumptions, construct a structured FPV property plan, implement the planned SVA properties after engineer approval, and diagnose failures after formal-tool execution. The agent was evaluated on multiple submodules of security IP designs using Synopsys VC Formal, indicating substantial reductions in estimated FPV engineering effort. To our best knowledge, prior publications have not addressed automated planning, implementation, and debug of formal properties within a unified workflow. In our work, the methodology has been applied with Synopsys VC Formal and it is applicable for Cadence JasperGold [5] as well.

2 Workflow

Figure 1 summarizes the proposed AI-assisted AEP and FPV workflow. The blue blocks denote manual effort in the traditional flow, whereas the proposed agent automates the principal specification, planning, implementation, debug, and reporting activities. Engineer oversight is retained at key checkpoints for reviewing and approving the agent’s outputs to ensure the correctness and completeness of the artifacts generated. This integrated structure reduces repetitive manual intervention, improves traceability across verification stages, and enables verification engineers to focus on validating design intent, constraints, generated properties, and diagnostic evidence. In addition, the workflow maintains explicit links among source requirements, planned properties, generated verification collateral, and formal results, thereby supporting systematic review and analysis throughout successive design iterations. The resulting division of responsibilities combines scalable

automation with the technical judgment required for the key checkpoints on the specification confirmation, constraint validation, test plan review, code authoring, and final verification signoff where it includes coverage assessment.

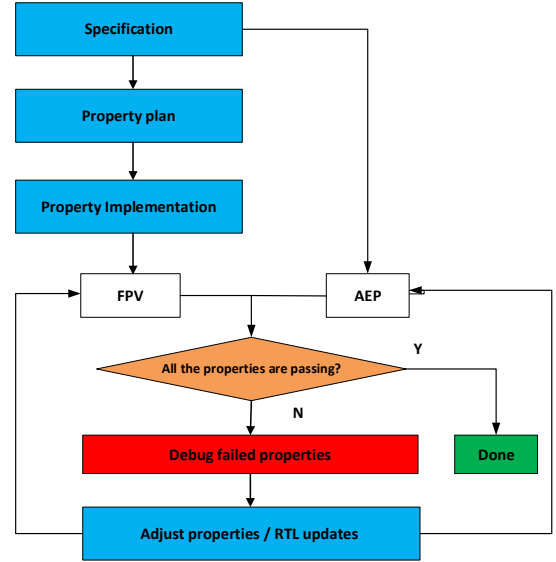


Figure 1. Proposed agent-assisted AEP and FPV workflow.

To run AEP and FPV separately, the agent executes the following six steps with user’s supervise:

- (1) specification and RTL analysis,
- (2) test-plan creation,
- (3) SVA-interface implementation, including modeling code and properties,
- (4) automated generation of comparison statistics,
- (5) Tcl-file generation, and
- (6) AEP-constraint generation.

All the steps are performed under verification-engineer supervision, with approval at the key checkpoints described above.

2.1 Specification Digest

Designers commonly develop RTL from high-level specifications that omit detailed implementation criteria. Microsoft Copilot [6] was used to extract relevant design details from the high-level specification document along with the recorded meeting transcripts containing discussions between designers and verification engineers.

By analyzing meeting transcripts together with the high-level design specification, the agent can construct a more complete description of the target block. It relates discussion context to high-level requirements and generates a block-level specification for designer review. During evaluation, this analysis also highlighted conditions requiring RTL review before detailed functional checking.

After confirmation, the block-level specification is used to derive candidate AEP constraints and generate the FPV test plan. The verification engineer should nevertheless begin with an unconstrained AEP run, which establishes a

baseline for identifying failures and evaluating whether subsequently introduced assumptions could over-constrain the design during the formal verification process.

2.2 AEP Constraint Generation

For failed AEP properties, the agent uses the AFi Model Context Protocol (MCP) server, developed in AMD internally, to identify relevant failure conditions and propose candidate constraints. These candidates are compared with assumptions derived from the specification. The verification engineer then determines whether an assumption is necessary, appropriate, and free from over-constraint, and identifies any omitted assumptions. It is worth noting that because high-level specifications and design discussions may be incomplete or ambiguous, manual review remains essential before constraints are applied.

2.3 AEP Failure Analysis and Debug

Although the principal AEP steps are similar with and without AI assistance, the MCP-assisted flow can reduce the manual effort required for failure analysis. Using the AFi MCP server, the agent locates relevant RTL logic, identifies conditions associated with a falsified property, and provides focused evidence to support root-cause analysis. Manual debug typically requires engineers to inspect multiple files and trace the relevant logic line by line. The agent-generated context therefore narrows the search space and can support more efficient convergence of the AEP task.

2.4 Property Plan Generation

Using the confirmed block-level specification and a predefined template, the agent generates a structured property plan. The template records the property name, a concise description, the property type (assumption, assertion, or cover), and a reference to the relevant specification section. This structure enables each planned property to be traced to its design intent before implementation. Designers and verification engineers review the plan to confirm that

- each property is necessary,
- all enabled features are checked by one or more properties,
- the planned assumptions are necessary and appropriate, and
- sufficient cover properties are included.

2.5 Property Implementation

After the property plan is reviewed by the designer and verification engineers, the agent implements the planned properties using a shared SVA rule file, which is another contribution of this work. This file is based on the “SVA Coding Guidelines” in Appendix B of [1] and includes additional recommended and discouraged examples. More than 14 rule categories promote consistent, tool-suitable SVA, including unique `S_/A_/C_` labels; default clocking and reset usage; simple, single-clock properties; bounded

temporal ranges; separation of assumptions and checkers; primary-input-only constraints; implication-free covers; and avoidance of implication chains, large `$past` delays, `first_match`, `until`, `until_with`, and `throughout`.

Figure 2 presents an example rule for simplifying generated SVA so that it aligns more effectively with the model-checking engines used by general-purpose formal verification tools. Owing to space limitations, the complete rule set is not presented in this paper.

```

141 ## 4. Temporal Operators and Structural Simplicity
142
143 ### 4.1 Keep Properties Simple
144
145 **Rule**: Decompose complex behaviors into multiple simple properties.
146
147 ```systemverilog
148 // BAD: Complex conjunction in consequent
149 A_complex: assert property(@(posedge clk) seq | => (p1 && p2 && p3));
150
151 // GOOD: Split into separate assertions (parallel proof, better constraint usability)
152 A_p1: assert property(@(posedge clk) seq | => p1);
153 A_p2: assert property(@(posedge clk) seq | => p2);
154 A_p3: assert property(@(posedge clk) seq | => p3);
155
156
157 **Benefits**:
158 - Faster compilation and proof
159 - Parallel verification
160 - Usable as constraints

```

Figure 2. An example of the rules for generating SVA code.

2.6 FPV Failure Analysis and Debug

As in the AEP flow, FPV failures are analyzed through the AFi MCP server, which connects the AI workflow to the formal verification infrastructure and support both VC Formal and JasperGold. After formal execution, the agent reviews the results provided by the formal tool, examines failed or inconclusive properties, correlates them with RTL logic and specification intent, and summarizes likely root causes. It also indicates whether a failure may originate from the RTL, an incorrectly generated assertion, a missing assumption, or a misunderstanding of the specification that requires confirmation against the source specifications and meeting records. In contrast to manual debug, which repeatedly examines RTL files, waveforms, logs, and potentially regression data, the MCP-assisted flow provides focused failure context and candidate corrections. These diagnoses and proposed changes are advisory; verification engineers review the evidence and approve any RTL, SVA, or constraint modification before it is applied.

3 Experiments and Results

The proposed agent was evaluated for both AEP and end-to-end FPV tasks across representative production-level semiconductor design blocks, spanning specification interpretation, property implementation, formal execution support, debug assistance, and result reporting. The evaluation measured the extent to which the conventional manual workflow could be automated while preserving engineer review at key decision points.

3.1 Evaluation Setup

The agent was implemented in an internal AI coding assistant integrated with Visual Studio Code 1.124.2 and configured with the anthropic:claude-sonnet-4-6-default:1m model. It operated as a multi-step reasoning system that

parsed specifications, generated property plans, implemented SVA interfaces, and prepared comparison reports without repeated per-query prompting. The experiments used Synopsys VC Formal and compared the generated results with existing formal verification tests. The flow is also applicable for Cadence JasperGold.

The evaluation focused on formal verification automation because the specification documents had already been confirmed during the corresponding manual verification efforts. For all effort measurements, one working day was defined as eight working hours. After generating each SVA interface, the agent produced a structured comparison report against the existing manual implementation. Specified-feature coverage was calculated as the percentage of intended features identified in the confirmed specification and represented in the approved property plan that were addressed by at least one implemented verification property. This automated comparative evaluation reduced manual post-analysis and improved evaluation reproducibility.

3.2 Evaluation Results

Tables 1 and 2 present detailed results for *module A*, a representative low-complexity design containing 180 bits of sequential SRAM, 13 flip-flops, two counters, a seven-state finite-state machine (FSM), six arithmetic operators, and 191 multiplexers.

Category	Existing (manual)	New (AI from spec)	Difference
ASSUME – unique named	20	15	-5
ASSERT (total)	13	13	0
COVER (total)	0	13	+13
TOTAL properties	33	41	+8

Table 1. Comparison of generated properties.

Metric	Existing (manual)	New (AI from spec)	Notes
Time required on property implementation	7 days	0.5 days	~93% reduction
Number of comment lines	~40	~60	New has per-property spec section references
Number of code lines	~150	~200	Comment line and blank line are not counted
Number of internal variables	12	4	Modelling signals
Number of always blocks	4	1	New uses single always block
Spec coverage (vs testplan)	~58%	100%	Existing misses all 13 covers
Property re-generation for design change	3 days	0.25 days	~92% reduction

Table 2. Comparison of efforts in formal verification.

As shown in Table 1, the AI agent eliminated five redundant assumption properties from both the property plan and the generated implementation. It also introduced **13 cover properties** targeting all seven FSM states and six key operational scenarios, thereby increasing specified-feature coverage from approximately **58% to 100%**. Although the resulting property set is larger, the additional cover properties provide explicit verification targets for assessing the coverage of the specified FSM states and operational scenarios. Concise comments also improve the readability of the generated code. As summarized in Table 2, the agent produced a more compact formal model by reducing the number of internal modeling variables from 12 to 4 and consolidating helper logic into fewer *always* blocks. These

structural reductions may reduce the effective state space presented to the formal solver and improve proof performance; however, solver-runtime effects were not measured separately in this evaluation. For module A, agent-assisted property planning and implementation required an estimated **0.5 working day (4 hours)** for one configuration. While, a conventional block-level FPV cycle, including formal execution, failure debug, review, and formal sign-off, required approximately **7 working days**.

Table 3 extends the comparison across the evaluated modules by summarizing property counts and implementation-effort metrics. For the first three modules, the agent reduced the number of assumptions by consolidating redundant assume properties and eliminating equivalent constraints expressed in different forms. Whereas cover properties were used less consistently in the manual implementations, the agent generated cover properties for all key scenarios, thereby providing explicit targets for assessing their coverage and improving verification-plan completeness.

Across all modules, agent-assisted property planning and implementation required an estimated 4 hours. Similar as the representative block, the conventional block-level FPV cycle requires approximately 56 hours for a configuration. But the verification cycle time would be reduced for a design applied in various configurations, as the agent generated files are configurable and the formal test of this design is reusable. If a design is changed, the estimated property-regeneration effort decreased from three working days (24 hours) to 2 hours, corresponding to a 12 $\times$ reduction in estimated engineering effort.

Table 3 also reports the number of internal variables used as modeling signals. Fewer internal variables may reduce the effective state space presented to the formal solver and improve proof performance. Finally, the agent consolidated helper logic into a single *always* block for each module. This simplification reduced the complexity of the combined formal model; solver-runtime effects were not measured separately in this evaluation.

Metric	Module A	Module B	Module C	Module D
A. Property Counts				
ASSUME – unique named	20 $\rightarrow$ 15	13 $\rightarrow$ 9	25 $\rightarrow$ 13	5 $\rightarrow$ 9
ASSERT (total)	13 $\rightarrow$ 13	13 $\rightarrow$ 14	27 $\rightarrow$ 13	11 $\rightarrow$ 13
COVER (total)	0 $\rightarrow$ 13	0 $\rightarrow$ 8	13 $\rightarrow$ 15	0 $\rightarrow$ 8
TOTAL properties	33 $\rightarrow$ 41	26 $\rightarrow$ 31	65 $\rightarrow$ 41	16 $\rightarrow$ 30
Spec coverage (vs testplan)	~58% $\rightarrow$ 100%	~61% $\rightarrow$ 100%	~43% $\rightarrow$ 98%	~53% $\rightarrow$ 100%
B. Effort Metrics				
Time required to create properties	7d $\rightarrow$ 0.5d	7d $\rightarrow$ 0.5d	7d $\rightarrow$ 0.5d	7d $\rightarrow$ 0.5d
Property re-generation (design change)	3d $\rightarrow$ 0.25d	3d $\rightarrow$ 0.25d	3d $\rightarrow$ 0.25d	3d $\rightarrow$ 0.25d
Internal variables (modeling signals)	12 $\rightarrow$ 4	5 $\rightarrow$ 2	18 $\rightarrow$ 4	16 $\rightarrow$ 4
Always blocks	4 $\rightarrow$ 1	2 $\rightarrow$ 1	3 $\rightarrow$ 1	1 $\rightarrow$ 1

Table 3. Overall results for four FPV experiments.

Note: In Table 3, “d” denotes an eight-hour working day; therefore, 0.25 d is equivalent to 2 hours.

Overall, the agent transforms FPV from manual property authoring and ad hoc debug into a supervised automation

flow that supports more systematic coverage planning and standardized reporting while reducing estimated engineering effort through planning, implementation, debug, and comparative analysis. As summarized in Table 3, the agent consistently generated more comprehensive property sets, particularly cover properties, across the evaluated experiments for assessing scenarios to be covered.

4 Conclusion

This paper presented an AI-agent framework that transforms natural-language design intent into a supervised, end-to-end AEP and FPV workflow encompassing specification extraction, AEP-constraint generation, property planning, SVA implementation, result comparison, and failure diagnosis. The evaluation summarized in Table 3 demonstrates the principal contributions of the proposed framework across four FPV experiments. For every evaluated module, agent-assisted property planning and implementation required an estimated **0.5 working day (2 hours)**. For context, the conventional block-level FPV cycle, including formal execution, failure debug, review, and sign-off, required approximately **7 working days (56 hours)**. Following a design change, the estimated property-regeneration effort decreased from **3 working days (24 hours)** to **2 hours**, corresponding to a $12\times$ reduction in estimated engineering effort. The agent also consolidated redundant assumptions for the first three modules (shown in Table 3), generated cover properties that provide explicit targets for assessing the coverage of key scenarios, reduced the number of internal modeling variables, and implemented helper logic in a single *always* block for each module. These results indicate that the framework supports more systematic verification planning, produces a more compact formal model, and reduces estimated engineering effort; the resulting solver-runtime effects were not measured.

The principal contribution is a practical agentic methodology that extends isolated prompt-based assistance into a reusable, specification-driven automation loop with engineer approval at key checkpoints. By integrating block-level specification generation, rule-guided SVA creation, automated comparison reporting, and MCP-assisted debug, the framework addresses both verification quality and workflow efficiency. The Table 3 results show that these gains are consistent across modules rather than limited to a single representative design: the agent produced more systematic property sets, particularly through cover-property generation, while reducing modeling complexity and estimated engineering effort for initial implementation and subsequent regeneration. The methodology is demonstrated using Synopsys VC Formal, but it is applicable in using Cadence JasperGold [5]. A remaining limitation is that extracted specifications may omit corner cases or constraint details. If such gaps are not corrected during engineer review, the generated test plan may omit relevant properties or introduce assumptions that over-

constrain the design; therefore, formal sign-off and expert review remain essential.

Future work will extend the agent beyond AEP and FPV to additional formal verification applications and provide the flexibility to switch among various formal verification tools, such as VC Formal and JasperGold, etc. One potential direction may be dynamically adjust orchestration setting to accelerate the convergent of the formal proof.

References

- [1] Synopsys, Inc., *VC Formal User Guide*, Version W-2026.03, Mountain View, CA, USA, Mar. 2026.
- [2] B. Sultan, L. Apvrille, and P. d. Saqui-Sannes, “Automated derivation of formal properties from requirements,” in *Proc. 2026 IEEE International Systems Conference (SysCon)*, Halifax, NS, Canada, 2026, pp. 1–8, doi: 10.1109/SysCon66367.2026.11503633.
- [3] A. Shambu, V. Dalal, B. Naik, and S. M., “Automation of FSM verification using formal tools,” in *Proc. 2023 International Conference on Communication, Circuits, and Systems (IC3S)*, Bhubaneswar, India, 2023, pp. 1–6, doi: 10.1109/IC3S57698.2023.10169571.
- [4] L. Liu, J. Parmar, N. Teggi, and S. Reddy, “Application of AI to accelerate formal verification workflow,” in *Proc. 2025 GSA Women’s Leadership Initiative Women in Semiconductor Hardware (WISH) Conference*, San Jose, CA, USA, Oct. 1, 2025. [Online]. Available: <https://designthesolution.org/wish-2025-final-papers/>. Accessed: Jul. 2026.
- [5] Cadence Design Systems, Inc., “Jasper Formal Verification Platform.” [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification.html. Accessed: Jul. 2026.
- [6] Microsoft, “Copilot.” [Online]. Available: <https://copilot.microsoft.com>. Accessed: Jul. 2026.