SIGMA: Sign-off Intelligence with GenAI for Methodical Assurance in Formal Verification

R C Sanjay Krushnan, Cadence Design Systems, India (rcsanjay@cadence.com)
Moola Jeevan Chaitanya Goud, Cadence Design Systems, India (jeevanm@cadence.com)
Sakthivel Ramaiah, Cadence Design Systems, India (sramaiah@cadence.com)
Erik Seligman, Cadence Design Systems, USA (seligman@cadence.com)

***Abstract*- Formal verification ensures the functional correctness of digital designs through exhaustive mathematical validation. However, tasks such as extracting the verification plan, setting up the environment, and developing formal claims pose significant challenges. These tasks demand substantial time and expertise, increasing the risk of missed checks, and reducing overall efficiency. This paper introduces a GenAI-Augmented Formal Verification Sign-off Flow designed to address these bottlenecks. The proposed solution integrates Generative AI, automation scripts, and Copilot components to collectively reduce manual effort and enhance coverage. By automating critical steps in the verification process, this flow boosts productivity and enables engineers to focus on complex verification tasks, ultimately supporting efficient and high-quality IP development.**

I. Introduction

As digital systems continue to evolve, ensuring their functional correctness has become a critical challenge in semiconductor design. Formal verification provides a mathematically rigorous approach to validate design behavior, offering exhaustive coverage that traditional simulation-based methods often lack.

Recent advancements in Generative AI and automation tools present new opportunities to streamline verification processes. This work proposes a Formal Verification Sign-Off Flow that integrates three automation-driven components:

1. **GenAI** – a platform that automatically extracts verification plans directly from design specifications.
2. **Python-based automation** – simplifies the creation of verification environments.
3. **AI Assistant** – assists in generating System Verilog Assertions (SVAs) using copilot.

By embedding intelligence into traditionally manual tasks, the proposed flow enhances verification efficiency and enables engineers to focus on higher-value analysis.

II. Motivation

In current Formal Verification Sign-off flows, several critical tasks are still performed manually, introducing the risk of human error and increasing verification turnaround time – leading to inefficiencies, increased verification effort, and potential coverage gaps.

Key Challenges in this process are:

1. Extracting the verification plan from the base and implementation specifications is challenging. It requires interpreting complex design intent and translating it into formal properties. Because this step depends heavily on expert judgment, it increases the chances of overlooked checks.
2. Creating formal environments involves instantiating DUTs, setting up constraints, and defining the necessary abstraction models. Since this setup is usually done manually, it demands considerable engineering effort and can slow down the overall verification cycle.
3. The development of System Verilog Assertions (SVAs) for basic functional checks is repetitive and time-consuming, limiting the bandwidth available for crafting high-value, protocol-specific properties.

These limitations underscore the need for an automated, intelligent formal verification flow that can (1) accurately derive verification plan from specification documents, (2) auto-generate formal environments with minimal user input, and (3) accelerate SVA creation for standard checks. Such a solution would enhance verification coverage and improve overall productivity in IP development.

III. METHODOLOGY

To improve the reliability and efficiency of Formal Verification Sign-Off by minimizing manual intervention, we propose a GenAI-Augmented Formal Verification Sign-Off Flow that integrates three core components: GenAI, Python-based automation, and Copilot. This architecture is designed to automate the most time-consuming and error-prone stages of the formal verification process while maintaining traceability, configurability, and human-in-the-loop validation.

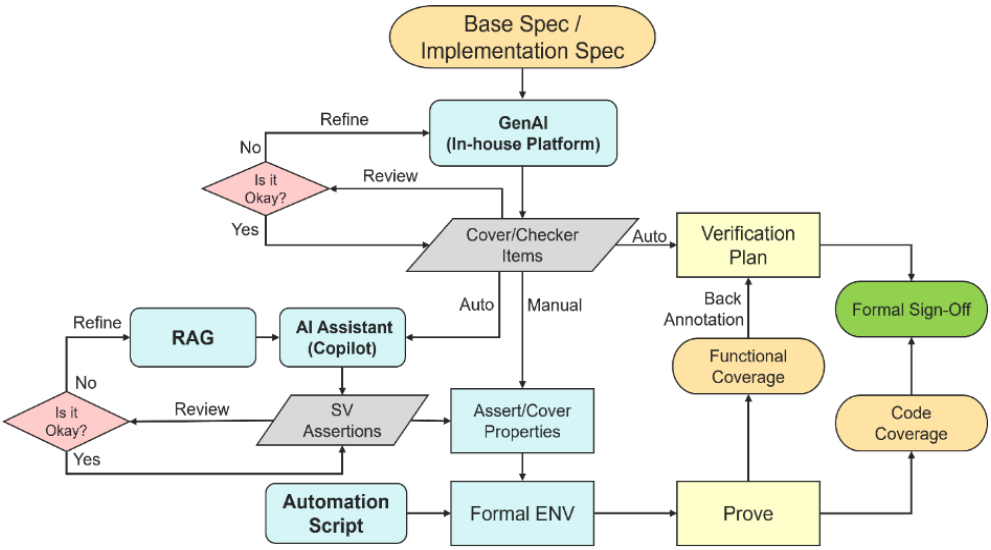

Figure 1. GenAI-Augmented FV Sign-Off Flow

A. GenAI: LLM-Driven Verification Plan Extraction and Classification

Generative AI platform employs a domain-adapted LLM to parse both architectural and micro-architectural specifications. Performs a semantic analysis to extract verification features and mapped to formal check categories. The extracted features are structured into a hierarchical verification plan, which is iteratively refined using prompt tuning and engineer feedback. GenAI also classifies checks into basic (e.g., signal stability, protocol compliance) and complex (e.g., multi-cycle dependencies, temporal ordering) categories, enabling downstream automation and prioritization.

a) Specification Ingestion and Verification Plan Generation:
   1. Input: Base/implementation specifications and reference Verification Plan format are fed into the in-house Generative AI platform.
   2. Process:
      - GenAI uses NLP and LLM-based understanding to extract potential verification checks.
      - These extracted checks are evaluated for quality and completeness.
      - Then, GenAI converts the verification checks into a standardized verification plan format with the help of reference template to ensure consistency and compatibility.

b) Steps to ensure the verification plan generated is reliable:
   1. Specification Quality Check
      - Pre-processing Validation: Ensure the Implementation specifications are complete, unambiguous, and well-structured before ingestion.
      - Semantic Consistency: Use NLP tools to detect inconsistencies, contradictions, or vague language in the specs.
   2. Multi-Pass Extraction and Cross-Referencing
      - Redundant Extraction Passes: Run multiple extraction passes using different LLM prompts or models to compare and consolidate verification checks.
      - Cross-Reference with Standards: Validate extracted checks against known protocol standards or industry best practices.

3. Human-in-the-Loop Review
    - Expert Review Layer: Involve domain experts to review and refine the generated checks before finalizing the plan.
    - Feedback Loop: Incorporate feedback from verification engineers to improve future generations.
4. Metrics and Scoring
    - Completeness Score: Rate the plan based on how many specification items are covered.
    - Quality Score: Evaluate clarity, redundancy, and relevance of each check using LLM-based scoring models.

B. Automation: Scalable Formal Environment Generation

The formal verification environment setup has been streamlined through an in-house developed Python-based automation script.

This script intelligently parses the Design Under Test (DUT) to extract essential elements such as clock, reset, parameters, inputs, and outputs. Leveraging this extracted data, it automatically generates a comprehensive formal verification environment. This includes the creation of a top-level module, a checker module, a *.f file that specifies the paths to both the RTL and the formal environment components, and a Makefile containing the necessary TCL commands.

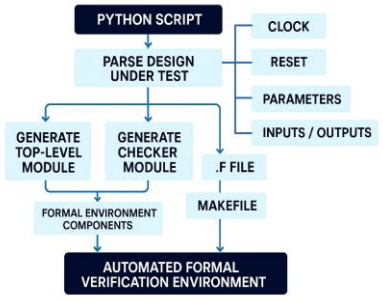

Figure 2. Illustration of Automation Script Framework

The automation ensures accurate port instantiation and correct bindings, reducing manual effort and human error. Parameterized scripts support multiple IP configurations and integrate seamlessly with formal tools. This script-driven flow accelerates setup, enforces consistency, and enables scalable, repeatable formal verification for multiple DUTs with minimal intervention. This approach is particularly suited for large-scale projects with extensive I/O interfaces, as it shifts the emphasis from setup logistics to verification strategy and coverage, thereby enhancing both productivity and overall quality. Additionally, it supports reproducibility, faster environment creation, and CI-based formal regressions.

C. AI Assistant: Assertion Generation and RAG-Based Refinement

The Proprietary AI Assistant, powered by Copilot, automatically generates System Verilog Assertions (SVAs) for all fundamental checks defined in the verification plan. These assertions are generated using rule-based templates and LLM-guided synthesis. For complex checks, the flow incorporates Retrieval-Augmented Generation (RAG), which retrieves relevant design context, historical assertions, and specification fragments to assist engineers in refining or constructing advanced properties. This hybrid approach balances automation with expert oversight, ensuring both correctness and completeness.

Steps to ensure Reliability of AI-Generated SVAs:

1. LLM-Guided Assertion Synthesis
    - Prompt Engineering: Use well-structured prompts to guide the LLM toward generating context-aware and logically sound assertions.
    - Model Confidence Scoring: Evaluate the confidence levels of generated assertions and flag low-confidence outputs for manual review.
2. Retrieval-Augmented Generation (RAG) Reliability
    - Context Relevance Filtering: Ensure retrieved design fragments, historical assertions, and spec snippets are contextually aligned with the current DUT.
    - Traceability Links: Maintain traceability between each generated assertion and its source context (spec fragment or historical reference).
3. Human-in-the-Loop Oversight
    - Engineer Review Portal: Provide a UI for engineers to review, edit, and approve assertions before integration.
    - Feedback Loop: Capture engineer feedback to refine future assertion generation and improve template/LLM behavior.

D. Formal Proof Execution

The complete environment is executed using a formal EDA tool, leveraging exhaustive state-space traversal to evaluate System Verilog Assertions (SVAs). Each property undergoes rigorous formal analysis to determine provability or falsification. For failing assertions, debug procedures include waveform inspection and counterexample tracing. Root cause analysis is conducted in conjunction with design teams to ensure resolution and closure of verification gaps.

E. Coverage Analysis and Sign-Off

Post-verification, Code coverage analysis is performed using formal EDA tool. Functional Reachability is measured using stimuli coverage. Checker completeness is measured using Cone of Influence (COI) and Proof Core. Functional coverage is analyzed after back-annotating coverage metrics to the verification plan, ensuring alignment with predefined verification objectives. Coverage metrics are compiled into traceability reports providing audit-ready documentation for formal sign-off and demonstrating verification completeness and correctness with quantifiable metrics.

TABLE I
COMPARISON BETWEEN TRADITIONAL VS SIGMA FLOW

Aspect	Traditional Sign-Off Flow	SIGMA Flow
Core Components	Manual processes, static checklists, engineer-driven	GenAI, automation scripts, Copilot components
Manual Effort	High; engineers manually manage sign-off steps	Reduced; automation handles critical steps
Coverage Improvement	Incremental, dependent on manual assertion and test creation	Enhanced through intelligent coverage analysis and GenAI suggestions
Productivity	Slower due to repetitive tasks and manual debug	Boosted by automation and intelligent assistance
Engineer Focus	Split between routine and complex tasks	Freed from routine tasks to focus on complex verification challenges
Verification Quality	Varies with team experience and effort	Consistently high due to methodical assurance and AI support
IP Development Impact	Longer cycles, delayed feedback	Efficient and high-quality IP development with early insights
Scalability Across Blocks	Requires manual adaptation and setup	Seamless deployment across blocks with reusable methodology
Outcome in Case Study	Baseline verification confidence	Facilitated Traditional flow, enabled left shift, adopted for future blocks

IV. CASE STUDY

To validate the efficacy of the SIGMA formal verification methodology, one of the PCIe functionality Sequence Number Handshake was selected as a representative design. This block underwent verification using both the conventional Formal Verification Sign-Off Flow and the SIGMA flow, enabling a direct comparison of verification efficiency, coverage, and overall effectiveness.

A. Design Overview:

The Sequence Number Synchronization State Machine ensures proper flit sequence number alignment during PCIe link initialization and recovery. It transitions through four main states: IDLE (no flit exchange when link is down or in recovery), IDLE Flit Handshake Phase (exchanging IDLE flits to establish link readiness), Sequence Number Handshake Phase (synchronizing sequence numbers using explicit flits and updating NEXT_TX_FLIT_SEQ_NUM), and Normal Flit Exchange Phase (regular flit transmission with sequence tracking and error detection). This design guarantees protocol compliance, supports error handling, and maintains data integrity across link states.

B. Experimentation:

a) **LLM Model Deployment in GenAI Platform**

- The GenAI Platform supports a variety of LLM models for feature extraction and conversion of specification data into formal verification constructs.
- From the available options, the top three LLM models were selected based on a balance of performance, cost, and efficiency.

- Despite potential cost trade-offs, the focus is on deploying robust models that help reduce verification timelines and catch bugs early, thereby minimizing overall development costs.
- In the SIGMA verification flow, the following LLM models have been deployed: GEMINI, GPT and Llama.

The feature extraction phase, which typically required approximately two weeks of manual effort, was completed in just three days using a GenAI platform powered by an LLM. This resulted in a 75% reduction in turnaround time. The LLM enabled automated parsing of specifications and generation of verification artifacts, significantly improving efficiency, consistency, and scalability in the formal verification planning process.

Query Prompt for Extracting Verification Checkers from Base/Implementation Specification

Formulate a formal verification strategy for the provided implementation specification, emphasizing the systematic extraction and definition of protocol-level checkers and functional coverage items. It must detail the construction of verification checkers targeting control signal sequencing, data integrity, and protocol compliance across all operational scenarios.

TABLE II
LLM MODEL COMPARISON

S.No	LLM Model	Completeness Score	Quality Score	Advantage	Examples
1	GEMINI	80%	90%	High accuracy; excels in hierarchical plan generation and complex check classification	**IDLE State** **Entry Condition:** Prove that the SM enters or remains in initial state when Link is down, or the link enters recovery. **No Transmission**: Prove that while in the initial state, no flits are transmitted. **Exit Condition:** Prove that if the SM is in initial state and ltssm enters L0 or Configuration.Idle or Recovery.Idle, the next state is IDLE FLIT Handshake PHASE.
2	GPT-OSS 120B	75%	80%	Moderate performance; suitable for general-purpose extraction tasks	**Idle Transition** If LTSSM enters L0 (or Configuration.Idle/Recovery.Idle) while in initial state, the SM moves to IDLE FLIT Handshake PHASE. **IDLE Flit Exchange** In IDLE FLIT Handshake PHASE, the transmitter must continuously send IDLE flits; any non-IDLE received flit is ignored.
3	Llama-3_3-nemotron-super-49b-v1_5	65%	72%	Efficient parsing of specifications; good semantic analysis for basic checks	**State Machine Transitions** **Entry to IDLE:** Verify on Link is down or LTSSM recovery entry. **IDLE to IDLE_FLIT_HS_PHASE:** Verify transition occurs when LTSSM enters L0/Configuration.Idle/Recovery.Idle.

Note: The above results obtained from various LLMs are specific to the given design. However, the quality and accuracy of these results may vary depending on the design's functionality and complexity.

b) **AI Assistant:** Spec-derived checks are converted into SystemVerilog Assertions by an AI assistant and refined for reuse via a Retrieval-Augmented Generation (RAG) framework. This automation reduced the overall timeline by approximately 1.5 weeks, significantly accelerating the verification process. An example of a feature is described below:

TABLE III
SVA COMPARISON

Manual SVA	(state == IDLE && (link_state_rec_idle || link_state_cfg_idle)) |=> (state == IDLE_FLIT_HS_PHASE)
AI Assistant SVA	(state == IDLE && (link_state_rec_idle || link_state_cfg_idle)) |-> (state == IDLE_FLIT_HS_PHASE)
RAG Refined SVA	(state == IDLE && (link_state_rec_idle || link_state_cfg_idle)) |=> (state == IDLE_FLIT_HS_PHASE)

As shown in Table III, the feature used to derive the SVA is "When the state is IDLE and either link_state_rec_idle or link_state_cfg_idle is asserted, the state transitions to IDLE_FLIT_HS_PHASE."

In the AI-assisted SVA verification flow, the design spec required a non-overlapping implication (|=>) to ensure a strict cycle transition from IDLE to IDLE_FLIT_HS_PHASE when either link_state_rec_idle or link_state_cfg_idle is asserted. Initially, the AI generated an assertion using overlapping implication (|->), allowing same-cycle transitions and violating protocol intent. The manually corrected assertion with |=> was stored via RAG, enabling the assistant to learn from feedback and regenerate assertions with accurate semantics. This refinement aligned with the spec and enabled successful verification. Similarly, RAG was utilized to manage a wide range of scenarios, including conditions involving $rose, $fell, custom signal names and values, as well as other complex syntax specific to this block.

c) **Automated Formal Environment:** It reduced setup time significantly—from approximately one week of manual effort to around two days—streamlining the verification process and improving overall efficiency**.**

C. Observations:

a) **SIGMA Flow Facilitated Traditional Flow:** The SIGMA methodology, powered by GenAI and automation, complemented the traditional flow by streamlining critical verification steps. This synergy helped validate the DUT more efficiently and reinforced the reliability of the traditional approach.

b) **Left Shift Achieved:** By automating routine tasks and enabling early bug detection, SIGMA allowed verification activities to begin earlier in the design cycle. This "left shift" reduced turnaround time by 1 month and improved overall development agility.

c) **Enhanced Engineer Productivity:** With automation handling repetitive tasks, engineers could focus on complex verification challenges. This led to faster debug cycles and higher-quality outcomes.

d) **Scalability Across Blocks:** The success of SIGMA in this DUT demonstrated its adaptability and scalability. Its methodology was reused for subsequent blocks, ensuring consistent verification standards and reduced setup effort.

e) **Confidence in Future Deployments:** The dual-flow verification of Seq Num Sync SM DUT built strong initial confidence. This justified the decision to adopt SIGMA as the primary flow for future IP blocks.

In this case study, verification was performed using both the traditional formal verification flow and the SIGMA flow to demonstrate that SIGMA delivers equivalent quality. This was achieved through a human-in-the-loop approach, where AI-assisted results were thoroughly reviewed to ensure complete and robust verification.

D. Bug Example:

Misinterpretation of Flit Type: During Flit Mode operation, a payload flit was incorrectly received when the flit usage field was set to 00, which is reserved for control flits such as NOP or IDLE. According to the protocol specification, payload flits must be accompanied by a flit usage value of 01, indicating valid data transmission. The anomaly occurred under a corner-case scenario triggered by a boundary condition, while normal operational flows remained unaffected.

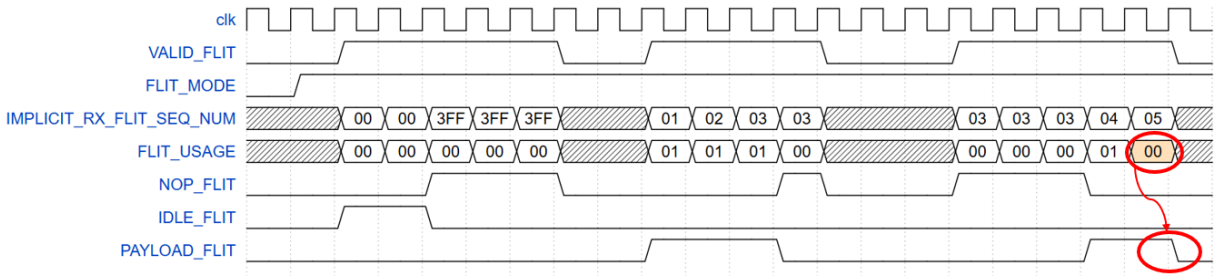

Figure 3. Illustration of Misinterpretation Flit Type

V. RESULTS

The proposed GenAI-augmented formal verification flow demonstrated significant improvements in both productivity and verification quality in real-world PCIe IP. By automating key stages—Verification Plan extraction, formal environment setup, and SVA generation—the flow enabled a measurable left shift in the development cycle, accelerating project timelines and reducing manual overhead. While the results were demonstrated on a PCIe-based IP, the approach is also broadly applicable to any design

The flow manages design complexity by leveraging LLMs and RAG to interpret intricate specifications, auto-generating modular formal environments tailored to the DUT structure, and synthesizing context-aware SVAs that scale across multi-domain, parameterized, and hierarchical designs—ensuring correctness through formal validation and expert oversight.

Following the promising outcomes from the initial block case study, the AI-assisted verification methodology was systematically applied to additional blocks using a human-in-the-loop approach for detailed analysis. Final investigations and bug resolutions were completed by the Verification Lead and the Designer, ensuring protocol compliance and design integrity.

A. Key enhancements observed include:

- Improved Accuracy: Automated Verification Plan extraction minimizes the risk of missed checks, ensuring comprehensive coverage of design intent.
- Increased Efficiency: Python-based automation eliminated repetitive setup tasks, reducing environment creation time by over 60%.
- Enhanced Coverage: Copilot-generated SVAs ensured that all checks were captured early in the cycle, improving formal convergence rates.

TABLE IV
FORMAL VERIFICATION TIMELINE

S. No	Block	Gate Count	Effort(months)		
			Pre-Project Estimation (Traditional)	Traditional FV Sign-Off Flow (Post-Project Estimation)	GenAI-Augmented FV Sign-Off Flow (SIGMA Flow)
1	SEQ NUM FSM	Low	4	4 (Actual)	3
2	ACK/NAK SCHEDULER	High	6	6	5
3	TX RETRY BUFFER	Medium	6	6	4.5
4	DLLP DECODER	Low	4	3	2
5	DLP INSERT	Medium	5	5	3.5

Gate Count: Low "<5000"; Medium ">5000 & <15000"; High ">15000"

Based on Seq Num Sync SM results, the SIGMA flow was extended to additional PCIe blocks. For these blocks, the reported SIGMA flow timelines correspond to actual implementation measurements. By contrast, the timelines for the traditional formal verification (FV) sign-off flow were extrapolated based on gate count and I/O count correlations derived from the SIGMA flow timelines, as applying an equivalent dual-flow verification methodology would resulted in prohibitive verification time. Nevertheless, full verification coverage was ensured through a human-in-the-loop review process.

B. Benefits:

The flow proved highly effective in onboarding junior formal verification engineers. With guided automation:

- Engineers quickly learned to extract features from specifications.
- They gained early exposure to SVA structure and application.
- Ramp-up time was significantly reduced, enabling meaningful contributions within the first few weeks.

Overall, the solution contributed to faster time to market, reduced verification costs, and higher design quality, validating its effectiveness as a scalable and intelligent Formal Verification Sign-Off methodology

VI. Future Scope

The current flow handles complexity, with RAG managing advanced checks. To further improve efficiency, we are developing automatic auxiliary code generation beyond assertions, which will eventually replace the AI Assistant while maintaining the same workflow and human oversight.

Future enhancements will focus on scalability and adaptability by:

- Integrating an Intelligent Assistant powered by a custom domain-specific language model.
- Using an IDE with modular code plugins for automated AUX code generation.
- Implementing AI Agent to assist debugging to find the root cause of the issue and suggest suitable bug fixes.

These advancements will deliver greater automation, deeper verification coverage, and faster onboarding of new engineers—positioning the methodology for long-term scalability and innovation.

VII. Summary

The GenAI-Enhanced Formal Verification Sign-Off methodology introduces a hybrid automation framework that blends AI-driven intelligence with engineer-guided validation to streamline and scale formal verification.

Basic checks are seamlessly transformed into System Verilog Assertions (SVAs) using AI Assistant, while complex properties are enhanced through Retrieval-Augmented Generation (RAG), leveraging contextual design knowledge for assertion refinement. The formal environment, including DUT instantiation, constraints, and assertion integration, is auto-generated via Python-based infrastructure, ensuring consistency, scalability and reduced manual effort.

To ensure verification completeness, the flow includes coverage closure mechanisms, allowing for measurable progress and traceability. This architecture not only reduces manual overhead but also accelerates onboarding, improves verification convergence, and ensures traceable, high-quality sign-off. The result is a scalable, intelligent verification framework optimized for modern IP development.

This methodology can be broadly adopted to verify any design blocks, making it a flexible and scalable solution for comprehensive formal verification across diverse protocol families.

References

[1] Erik Seligman, Tom Schubert, and M V Achutha Kiran Kumar. *Formal Verification, An Essential Toolkit for Modern VLSI Design*. Morgan Kaufmann Publishers, 2nd Edition, 2023.

[2] J. Deepak Narayan Gadde, Aman Kumar, Thomas Nalapat, Evgenii Rezunov, Fabio Cappellini. “All Artificial, Less Intelligence: GenAI through the Lens of Formal Verification”. In: DVCon US, 2024.

[3] I.S. Hafiz Abdul Quddus, Md Sanowar Hossain, Ziya Cevahir, Alexander Jesser, Md Nur Amin.”Enhanced-VLSI-Assertion-Generation-Conforming-to-HighLevel-Specifications-and-Reducing-LLM-Hallucinations-with-RAG”. In: DVCon Europe, 2024.