To appear at the 38th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI),
August 25-29, 2025, Manaus, BRAZIL

Hey AI, Generate Me a Hardware Code! Agentic AI-based Hardware Design & Verification

Deepak Narayan Gadde[1], Keerthan Kopparam Radhakrishna[1], Vaisakh Naduvodi Viswambharan[1],
Aman Kumar[2], Djones Lettnin [3], Wolfgang Kunz[4], Sebastian Simon[1]
[1]Infineon Technologies Dresden GmbH & Co. KG, Germany
[2]Infineon Technologies India Pvt. Ltd., India
[3]Infineon Technologies AG, Germany
[4]Rheinland-Pfälzische Technische Universität Kaiserslautern-Landau, Germany

***Abstract*—Modern Integrated Circuits (ICs) are becoming increasingly complex, and so is their development process. Hardware design verification entails a methodical and disciplined approach to the planning, development, execution, and sign-off of functionally correct hardware designs. This tedious process requires significant effort and time to ensure a bug-free tape-out. The field of Natural Language Processing has undergone a significant transformation with the advent of Large Language Models (LLMs). These powerful models, often referred to as Generative AI (GenAI), have revolutionized how machines understand and generate human language, enabling unprecedented advancements in a wide array of applications, including hardware design verification. This paper presents an agentic AI-based approach to hardware design verification, which empowers AI agents, in collaboration with Human-in-the-Loop (HITL) intervention, to engage in a more dynamic, iterative, and self-reflective process, ultimately performing end-to-end hardware design and verification. This methodology is evaluated on five open-source designs, achieving over $95\,\%$ coverage with reduced verification time while demonstrating superior performance, adaptability, and configurability.**

***Index Terms*—Agentic AI, LLM, Formal Verification, Hardware Design**

I. Introduction

The swift progress in semiconductor industry has enabled the creation of intricate IC designs with multiple functions on a single chip. However, this complexity has turned Register Transfer Level (RTL) design and verification into significant bottlenecks in IC development. RTL design requires extensive manual expertise to ensure functional accuracy, optimal performance, and power efficiency whereas, verification stage alone can consume up to $60\,\%$ of the total project time [1]. Even with advancements in EDA tools, verification inefficiencies persist, leading to increased costs and time-to-market.

Simulation-based verification, the industry standard, offers high coverage but is computationally expensive and time-consuming. As IC complexity scales, exhaustive state-space validation becomes impractical due to exponential verification effort. Formal verification, while mathematically rigorous, suffers from scalability limitations arising from manual property specification, model complexity, and challenges in verifying deeply sequential designs. These constraints have driven the search for AI-driven verification methodologies that enhance efficiency and throughput while reducing manual effort.

Recent advances in GenAI and LLMs have demonstrated potential in IC design and verification [2], enabling tasks such as RTL generation [3], testbench creation [4], stimuli generation [5], coverage closure [6], formal property generation [7], and vulnerability detection [8]. Frameworks like AIVRIL [3] and VeriAssist [4] leverage LLMs for RTL design but face scalability challenges due to their dependence on simulation. Moreover, applying LLMs to hardware design poses significant difficulties, as studies [9] report around $60\,\%$ failure rates in generated RTL designs due to issues like randomness, hallucinations, and challenges in handling complex specifications. These findings highlight the overly optimistic expectations of zero-shot LLM applications in hardware design tasks. Moreover, existing approaches often focus on static code generation but overlook iterative refinement, self-correction, and HITL integration, which are essential for reliable design and verification. To address these limitations, this work proposes a methodology based on Agentic AI, leveraging a Multi-Agent System (MAS) [10] to integrate autonomous AI agents for RTL generation and formal verification. Through MAS-driven adaptability, agentic coordination, and HITL-enhanced verification, this approach establishes a scalable, intelligent, and high-confidence verification paradigm for next-generation IC design.

The key contributions of this work are sketched as follows:

- Unlike conventional LLM-based approaches such as zero-shot, our work proposes MAS which enables collaborative agent-based code generation, iterative RTL refinement, dynamic property generation
- Fully autonomous end-to-end workflow where specialized agents can not only generate code but also interact directly with industry-standard EDA tools
- HITL integration to resolve ambiguities in AI-generated RTL and formal properties

II. Agentic Workflows

Traditional RTL design and verification workflows are predominantly non-agentic, relying on sequential, manual processes automated by scripting and EDA tools. In non-agentic workflows, engineers manually write RTL code or use template-based code generators, followed by the separate development of testbenches and formal properties. In contrast, agentic workflows distribute tasks among autonomous AI agents that coordinate to achieve a common goal. Rather than a single LLM executing prompts in isolation, multiple specialized agents

each equipped with distinct capabilities collaborate through structured interaction patterns, as shown in Fig. 1. The most prominent agentic design patterns include:

(a) Pipeline decomposition, which involves breaking down larger tasks into smaller actionable subtasks that agents can execute independently [11]

(b) Deliberation loops, in which agents iteratively propose designs or properties and critique one another to converge on high-quality outputs making final output more robust [10]

(c) Reflection and self-correction, enabling agents to review previous outputs, identify errors, and autonomously initiate re-generation enhancing reliability [12]

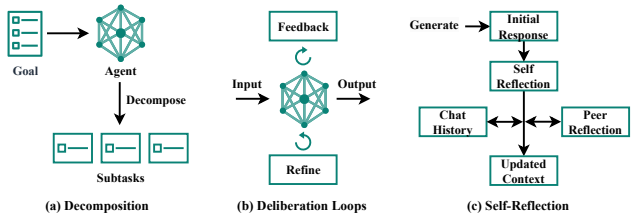

Fig. 1. Agentic design patterns

By enabling agents to think, research, generate, and review designs before producing final outputs, agentic workflows significantly improve reliability compared to zero-shot approaches while addressing key challenges including explainability of decisions, error propagation prevention, seamless EDA tool integration, appropriate human oversight, and coding standards compliance [13]. These workflows are orchestrated via multi-agent collaboration topologies ranging from broadcast group chats to hierarchical or role-based channels that assign tasks to specialized sub-agents, enabling dynamic information routing, peer review among agents, and efficient coordination that improves scalability, reduces redundancy, and fosters specialized expertise while maintaining HITL intervention for critical decision points.

III. Methodology

The proposed methodology leverages MAS paradigm to automate RTL design and verification flow integrating HITL interventions to ensure reliability and compliance. This approach is illustrated in Fig. 2. At its core, the system employs specialized AI agents that collaborate to transform high-level specifications into verified RTL implementations, balancing autonomous generation with human guidance. The process is structured into three distinct phases: planning, development, and execution. This structure ensures alignment with original requirements while facilitating iterative feedback and escalation when necessary.

The process begins with a structured specification document encompassing natural-language requirement descriptions, interface definitions, performance targets, and FSM details. In the planning phase, the *design lead* agent parses the specification to produce a microarchitecture detailing datapath components, control-state machines, reset strategies, timing constraints and more in a structured format. Concurrently, the *formal verification lead* agent interprets the same specification to generate a verification plan (vPlan), outlining property types, coverage goals and more. By deriving both the flows from a unified specification, any divergence between design intent and verification objectives are eliminated from the beginning.

During the development phase, task dispatchers channel the outputs from the planning phase into two parallel streams. In the design stream, RTL agents utilize the microarchitectural guidelines to generate SystemVerilog modules for each functional block, complying to coding standards and macro templates provided. In the verification stream, formal verification agents translate each vPlan entry into SystemVerilog Assertions (SVAs). Throughout this phase, critic agents monitor outputs flagging issues such as missing logic, syntax errors, coverage gaps and more. If predefined iteration limits are reached without resolution, a *conversable agent* escalates the issue to human reviewers for HITL intervention, addressing ambiguities beyond the capabilities of automated agents.

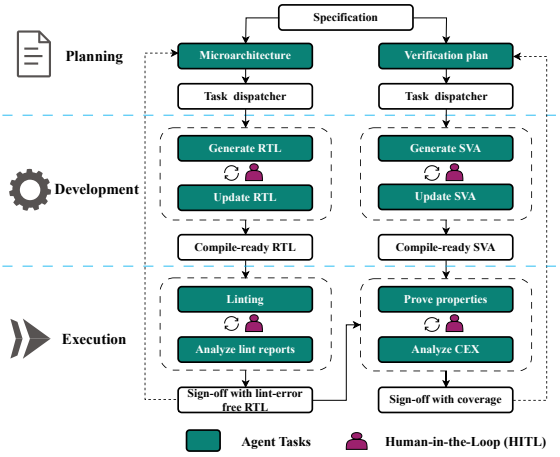

Fig. 2. Agentic AI methodology for RTL design and verification with HITL

In the execution phase, the code executor agent interfaces with industry-standard EDA tools to validate the generated RTL and SVAs. A dedicated *code executor agent* invokes linting tools such as SpyGlass [14] to ensure no lint errors are present. Lint errors are parsed, categorized, and returned to the RTL agents for automated patching or routed to HITL when errors exceed the agents repair capabilities. For formal verification, the *code executor agent* drives formal tools such as JasperGold [15] to prove each SVA. Counter Examples (CEXs) are analyzed, prompting iterative fixes by development agents or escalation to human engineers for resolution. A dedicated coverage agent consolidates code coverage, assertion coverage, and functional coverage metrics against the original vPlan targets. If coverage thresholds are unmet, the system iteratively generates additional properties to achieve defined goals. The methodology concludes with a formal sign-off, delivering a compile-ready RTL along

with a structured coverage report detailing proven properties, coverage results, and any exceptions.

A. Multi-Agent Framework Architecture

The proposed methodology employs a modular agent orchestration system designed to be compatible with open-source MAS including CrewAI [16], AutoGen [17], and LangGraph [18] while the current evaluation focus exclusively on Autogen. It facilitates scalable MAS through event-driven model enabling deterministic agent interactions which manages agent roles, responsibilities, and interaction patterns, enabling dynamic adaptation to diverse design requirements. The framework supports seamless integration of various LLMs including GPT-4o [19] and Llama3.1 [20] through a standardized API layer. This modularity allows for hot-swapping of additional models, agents, tools with minimal code modifications ensuring adaptability to evolving design requirements. Each agent within the system is assigned with specific roles which then coordinates with other agents through structured protocols.

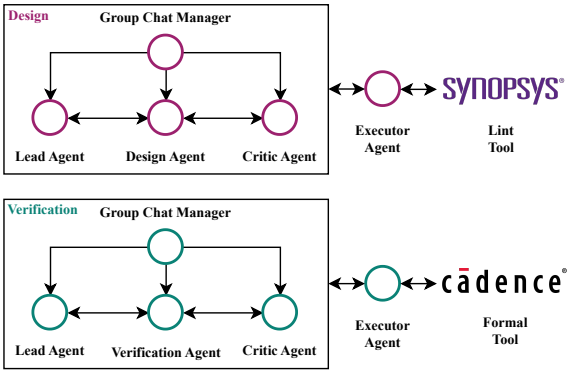

Fig. 3. Illustration of agentic AI workflow execution

Within the framework, AI agents collaborate to execute tasks specified by human users within a multi-agent group chat environment as shown in Fig. 3. Each agent specializes in specific roles such as *verification lead*, *formal verification engineer*, and *SystemVerilog LRM expert*. This design pattern enables agents to share a common message thread by subscribing and publishing to the same topic. Additionally, a HITL agent represents the human user, providing oversight and guidance when necessary to ensure the verification process aligns with human expertise and decision-making. The *Group Chat Manager* agent orchestrates the sequential interaction, ensuring that only one agent communicates at a time using AutoGen's Core API and an event-driven architecture.

To address common limitations of LLMs such as attention deficits, hallucination, and getting stuck in iterative loops, the methodology decomposes tasks into smaller tasks distributed among multiple agents. Throughout this process, critic agents evaluate the generated properties, providing feedback to enhance the quality and correctness of generated SVAs. A feedback loop with a threshold of five iterations is established; exceeding this limit triggers human intervention to prevent workflow stagnation and reduce hallucination errors.

The system incorporates comprehensive logging mechanisms that capture detailed interactions at multiple granularity levels including agent group chats, tool integration mechanisms, and error management systems. An integrated exception management system and threshold-based monitoring prevent infinite loops, automatically triggering human intervention when autonomous resolution is unattainable. Through this architecture, the methodology establishes a scalable, intelligent, and high-confidence verification paradigm for next-generation IC design, leveraging the strengths of both automated AI agents and strategic human oversight.

IV. Evaluation

The effectiveness of the proposed MAS methodology is evaluated by applying it to a range of internal and open-source design specifications. For benchmarking, five open-source designs—CRC, ECC, FIFO, Lemming, and Timer—were utilized. The specifications for CRC, ECC, and FIFO were sourced from OpenCores [21], while those for Lemming and Timer were obtained from VerilogEval [22]. The framework employs OpenAI's GPT-4o model, a 1.8 trillion parameter LLM, to generate and verify the RTL. To evaluate hyperparameter effects, the methodology was run at *temperature (temp)* of 0.2, 0.5, and 0.8, where lower values yield consistent results and higher values foster diverse output.

The RTL generation phase was evaluated using quality metrics, including lint error counts, logical correctness (classified as correct, correct but non-synthesizable, incorrect, or incomplete), and the time required for HITL intervention to achieve reliable RTL. For the formal verification phase, metrics such as the number of generated properties, achieved formal coverage percentage, CEX count, and additional HITL effort needed to approach near 100 % coverage were assessed. The HITL property count represents the final set of properties after redundancies are removed and new properties are added. The comprehensive results, presented in Table I, demonstrate the framework's effectiveness across designs of varying complexity, providing insights into the relationship between *temp* settings and design quality.

TABLE I
Overview of the results

Design	Temp	RTL Generation				Formal Property Generation					
		#Errors (linting)		Logical Accuracy		#Properties		Coverage (%)		#CEXs	
		MAS	HITL	MAS	HITL	MAS	HITL	MAS	HITL	MAS	HITL
CRC[21]	0.2	0	0	✔	✔	16	11	73.08	100	8	0
	0.5	0	0	!	✔						
	0.8	0	0	✖	✔						
ECC[21]	0.2	5	0	⊘	✔	19	12	93.88	95.90*	8	0
	0.5	2ϟ	0	⊘	✔						
	0.8	2ϟ	0	⊘	✔						
FIFO[21]	0.2	0	0	✔	✔	20	13	91.67	97.29*	11	0
	0.5	0	0	✖	✔						
	0.8	0	0	✔	✔						
Lemming[22]	0.2	0	0	✖	✔	18	21	96.15	96.77*	3	0
	0.5	0	0	✖	✔						
	0.8	1ϟ	0	!	✔						
Timer[22]	0.2	0	0	!	✔	42	57	83.95	96.39*	22	0
	0.5	0	0	✖	✔						
	0.8	0	0	✖	✔						

ϟ: Fatal, ✔: Correct, !: Non-synthesizable, ✖: Incorrect, ⊘: Incomplete.
The HITL effort for RTL and Formal Property Generation was 15/40 minutes for CRC, 30/20 minutes for ECC, 15/20 minutes for FIFO, 15/15 minutes for Lemming and 15/30 minutes for Timer, respectively.
* Indicates the uncovered portion is unreachable dead code

A. RTL Generation

The evaluation demonstrated that generated RTL for single module designs like CRC showed no lint errors across *temp*. Higher *temp* introduced randomness, occasionally producing incorrect RTL, as shown in Table I. Brief HITL interventions, typically under $15\,\mathrm{min}$, refined the RTL for better usability. For ECC design at 0.3 *temp*, two modules emerged with functional placeholders, causing lint errors. Specification refinement and adjusted agent configurations resolved these issues. Subsequent HITL efforts under $30\,\mathrm{min}$ completed missing logic. Higher *temp* produced severe lint errors, promptly addressed through targeted HITL. FIFO exhibited lint-free RTL across varying *temp*, demonstrating robust quality. At *temp* 0.5, a minor logical error was resolved with $15\,\mathrm{min}$ of HITL effort. Lemming passed linting for all *temp* values except 0.8, where a syntax error was identified. Functional bugs, common across all *temp* values, were resolved with HITL efforts, making the RTL ready for formal verification. Similarly, Timer designs passed linting at all *temp* values, but control logic bugs were present across all cases and were addressed through targeted HITL intervention.

The RTL generation process required varying numbers of iterative refinement cycles depending on design complexity. The Timer design converged after just 1 iteration, while CRC, FIFO, and Lemming designs required 2 iterations. The more complex ECC design, however, needed 3 iterations to achieve stable, functional RTL. The nature of human interventions during RTL generation primarily involved removing placeholder code segments and providing design-specific feedback to guide the agents. This iterative feedback loop improved the accuracy of the generated RTL while minimizing the need for extensive human rewrites.

B. Formal Property Generation

Following the lint-error free RTL generation, the verification flow generates formal properties, which are then validated and analyzed for coverage using formal tool, as detailed in Table I. The MAS approach achieved an average initial coverage of $86.21\,\%$ across all designs, with generated properties demonstrating high compilation success rates. Strategic HITL interventions, averaging $27\,\mathrm{min}$ per design, refined the properties to achieve $100\,\%$ assertion pass rate and an average final coverage of $97.73\,\%$, with remaining uncovered portions primarily consisting of unreachable dead code and default case statements.

The comparative analysis presented in Fig. 4 demonstrates that while zero-shot approaches achieved modest initial coverage averaging $69.85\,\%$, they required extensive prompt adjustments and iterative refinements even to ensure property compilation. In contrast, the proposed MAS approach demonstrated superior baseline performance with minimal iterations and, with HITL integration, achieved $100\,\%$ assertional pass rate and nearly $98\,\%$ coverage, demonstrating the effectiveness of the collaborative agent-human verification paradigm.

The human interventions during property generation focused primarily on addressing conceptual issues through targeted feedback and removing redundant properties. This complementary human-AI workflow capitalized on the agents' comprehensive property generation capabilities while leveraging human expertise to optimize verification coverage, resulting in higher quality properties and more efficient verification processes.

C. Discussion

The proposed methodology effectively manages complex design and verification tasks through specialized agents and process decomposition, addressing common LLM limitations. In contrast to zero-shot approach, MAS facilitates dynamic reasoning and iterative refinement where HITL interventions enhance robustness by resolving LLM errors and incomplete outputs, enabling accurate and complete RTL generation and verification through human-AI collaboration. The effectiveness of code generation heavily depends on the underlying quality of LLMs used, precise agent configurations through prompt engineering, and careful tuning of hyper parameters. While critic agents ensure the quality and correctness of generated properties through continuous feedback, their performance is contingent upon optimal agent count and configuration. Nevertheless, when properly configured, the framework's systematic task division, agent specialization, and HITL ensure comprehensive coverage, error-free RTL generation, and reliable verification.

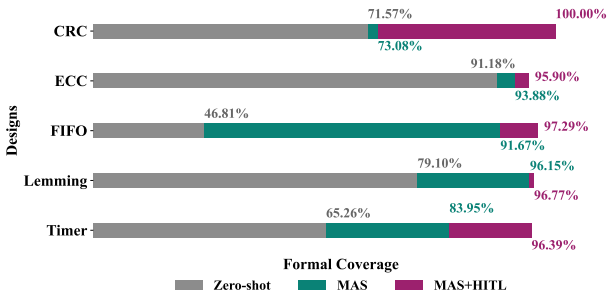

Fig. 4. Comparison of coverage results produced by proposed and zero-shot approaches

The HITL intervention model introduces an interesting cognitive trade-off. While human intervention times were relatively short (under 40 minutes), the cognitive effort required to analyze AI-generated designs may differ from reviewing human-created designs. The intervention process requires engineers to understand not only the design intent but also the reasoning patterns and potential limitations of the AI agents. However, the targeted nature of the interventions—focused on specific placeholders, lint errors, or counterexamples—reduces the cognitive load compared to comprehensive manual reviews.

V. Conclusion

This paper presents a novel MAS-based methodology for automated RTL design and verification, achieving around $95\,\%$ coverage through collaborative specialized agents and targeted HITL interventions. The approach demonstrates superior performance compared to zero-shot methods across various designs,

with MAS+HITL achieving up to $100\,\%$ coverage while requiring minimal human effort. Although not guaranteeing $100\,\%$ efficacy in every run, the methodology consistently demonstrates high-quality results, with robust performance influenced by hyper parameter settings and agent configurations. The hot-pluggable domain-agnostic architecture facilitates integration of emerging models, enabling adaptation to advances in LLM models, while the scalability analysis suggests applicability to more complex designs. Future work would focus on enhanced agent coordination, deeper EDA tool integration, explainable AI techniques to reduce human intervention advancing the state of AI-driven hardware design and verification towards truly autonomous systems. Furthermore, the methodology is being expanded to generate efficient testbenches and stimuli that addresses the verification of more intricate designs.

REFERENCES

[1] H. Foster, *Wilson Research Group IC/ASIC functional verification trend report*, Siemens Blog, 2024.

[2] M. Abdollahi *et al.*, *Hardware Design and Verification with Large Language Models: A Literature Survey, Challenges, and Open Issues*, MDPI Electronics Journal, 2024.

[3] M. ul Islam *et al.*, *AIvril: AI-Driven RTL Generation With Verification In-The-Loop*, arXiv, 2024.

[4] H. Huang *et al.*, *Towards LLM-Powered Verilog RTL Assistant: Self-Verification and Self-Correction*, arXiv, 2024.

[5] Z. Zhang *et al.*, *LLM4DV: Using Large Language Models for Hardware Test Stimuli Generation*, arXiv, 2023.

[6] C. Xiao *et al.*, *LLM-based Processor Verification: A Case Study for Neuromorphic Processor*, DATE, 2024.

[7] W. Fang *et al.*, *AssertLLM: Generating and Evaluating Hardware Verification Assertions from Design Specifications via Multi-LLMs*, arXiv, 2024.

[8] M. Akyash *et al.*, *Self-HWDebug: Automation of LLM Self-Instructing for Hardware Security Verification*, arXiv, 2024.

[9] D. N. Gadde *et al.*, *All artificial, less intelligence: Genai through the lens of formal verification*, 2024.

[10] S. Chen *et al.*, *A Survey on LLM-based Multi-Agent System: Recent Advances and New Frontiers in Application*, 2025.

[11] S. K. Jeyakumar *et al.*, "Advancing agentic systems: Dynamic task decomposition, tool integration and evaluation using novel metrics and dataset," 2024.

[12] M. Renze *et al.*, "Self-reflection in large language model agents: Effects on problem-solving performance," in *2024 2nd International Conference on Foundation and Large Language Models (FLLM)*, 2024, pp. 516–525. DOI: 10.1109/FLLM63129.2024.10852426.

[13] A. Patra *et al.*, *Aieda: Agentic ai design framework for digital asic system design*, 2024.

[14] Synopsys, *SpyGlass Lint*. [Online]. Available: https://www.synopsys.com/verification/static-and-formal-verification/spyglass/spyglass-lint.html.

[15] Cadence, *Cadence Jasper*. [Online]. Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification.html.

[16] CrewAI, *CrewAI-Cutting-edge framework for orchestrating role-playing AI agents*, 2024.

[17] Microsoft Research, *AutoGen: Framework for Multi-Agent Systems*, 2024.

[18] LangChain, *LangGraph - Build Stateful, Multi-Actor Applications*, 2024.

[19] OpenAI, *Chatgpt-4o*, 2025. [Online]. Available: https://openai.com/.

[20] Meta, *Llama*, 2025. [Online]. Available: https://www.llama.com/.

[21] OpenCores, *Open Source IP-Cores*, 2025. [Online]. Available: https://opencores.org/.

[22] N. Pinckney *et al.*, *Revisiting VerilogEval: Newer LLMs, In-Context Learning, and Specification-to-RTL Tasks*, arXiv, 2024. [Online]. Available: https://github.com/NVlabs/verilog-eval.