# Enhanced VLSI Assertion Generation: Conforming to High-Level Specifications and Reducing LLM Hallucinations with RAG

Hafiz Abdul Quddus*, Md Sanowar Hossain, Ziya Cevahir, Alexander Jesser, Md Nur Amin
*Institute for Intelligent Cyber-Physical Systems (ICPS)*
*Heilbronn University of Applied Sciences*
Künzelsau, Germany
{hafiz-abdul.quddus,md-sanowar.hossain, ziya.cevahir, alexander.jesser, md-nur.amin}@hs-heilbronn.de

***Abstract*—Assertion-based verification (ABV) is widely used in VLSI design. However, manual assertion writing is time-consuming and may not adhere to high-level specifications. Generative AI techniques like LLMs automate this but can introduce hallucination. We propose an automatic assertion generation framework using Retrieval-Augmented Generation (RAG) and LLMs. It generates assertions from designer-tailored specifications, ensuring conformance with high-level specifications and reducing hallucinations. We applied this to an AXI4-Lite protocol case study, verifying SystemVerilog Assertions (SVAs) against golden RTL using Bounded Model Checking (BMC). Results showed improved accuracy, conformance, and integration with ABV.**

***Index Terms*—Automated-Assertion Generation, Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), High-Level Specifications (HLS), Designer-tailored Specifications, SystemVerilog Assertions (SVAs), VLSI Design Flow, Bounded Model Checker (BMC).**

## I. Introduction

The Very Large Scale Integration (VLSI) design flow is pivotal for the development of complex chips, offering a structured view of design stages and Electronic Design Automation (EDA) tools to stakeholders [1]. The process starts with collecting customer requirements and system constraints by the system architect to prepare a high-level specification (HLS) document(s), typically in natural language. The designer interprets this HLS document to develop the RTL design. Additionally, the HLS serves as the golden specification in the verification flow. However, the ambiguity inherent in the natural language of HLS can lead to misunderstandings and potential bugs. The combined design and verification flow, shown in Fig 1, illustrates that the understanding gap between the designer and architect can introduce errors in subsequent stages, such as RTL to Netlist and physical layout translations. There can be several industry standards for specification analysis and languages to fill the gap in understanding [2]. However, design implementation conforming to the HLS remains an open challenge.

This work was funded by the BMBF of Germany (Grant 16ME0783) under the Ki4BoardNet project. See https://www.edacentrum.de/ki4boardnet/. The authors are solely responsible for the content.

*Corresponding Author
Implementation: https://github.com/HafizAQ/RAG-LLM-SPC-SVA

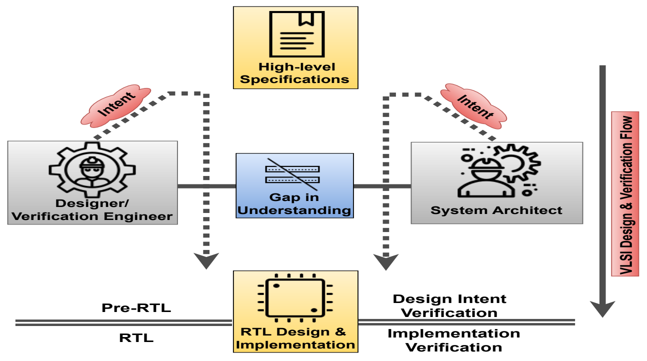
High-level Specifications
Intent
Intent
Designer/ Verification Engineer
Gap in Understanding
System Architect
VLSI Design & Verification Flow
Pre-RTL
RTL
RTL Design & Implementation
Design Intent Verification
Implementation Verification

Fig. 1. Understanding Gap with VLSI Design & Verification Flow: Specification-Centric Architect vs. Implementation-Centric Designer

Assertion-based verification (ABV) has emerged as a powerful technique that enhances controllability and observability in the design, making it highly effective for functional verification and bug detection in VLSI [1]. Similarly, both simulation and formal design validation settings can benefit from the application of ABV. This paper focuses on formal ABV, where no testbench is needed. To explore the RTL design state space against predefined assertions, ABV employs Formal Property Verification (FPV) with model checking [3]. Moreover, ABV can be applied at both pre-RTL and RTL stages for verifying design intent and implementation, as shown in Fig. 1. Under ABV, the designers-tailored specification sentences are translated into assertions to verify the design intents and RTL implementation processes. However, manually writing assertions is time-consuming and error-prone. Even with automation, incorrectly defined assertions can lead to false positives in design verification [4]. Although generative AI, such as LLMs, can generate assertions from design specifications, it can be hallucinations [5].

We summarize the challenges in two ways: first, assertion generation on the designer side should be consistent with the HLS document to bridge the gap of understanding between architects and designers, and second, to address the hallucination in automated assertion generation using LLMs. This paper addresses these challenges by combining transformer-based

LLMs with the NLP approach as Retrieval-Augmented Generation (RAG) for assertion generation. Our framework leverages RAG to store and retrieve HLS document(s) contextually and consult them for conformity. HLS documents, as RAG's contextual knowledge, are used with LLM's expressiveness to automate assertion generation. They establish conformance with HLS and reduce hallucination by semantically providing contextual awareness. We implement the proposed approach to the open-source AXI4-Lite protocol and evaluate it with a bounded model checker (BMC). The results demonstrate the effectiveness of our strategy in generating quality assertions consistent with HLS, reducing LLM hallucinations, and enhancing VLSI design verification.

The rest of the paper is structured as follows: Section II discusses related work on automated assertion generation, highlighting the research gap. Section III details the proposed approach, while Section IV describes the experimental setup, implementation, and evaluation. Section V presents the various aspects of our approach relevant to the EDA community. Finally, Section VI concludes with potential future work.

## II. Related Work

Poorly written assertions can deceive the ABV process, requiring high-quality assertion writing [4]. Assertion quality can be determined by its coverage through the design corner cases and how well it conforms to its specification [4]. Aligning verification with high-level specifications is also critical [2].

To address these challenges, the literature highlights three main approaches for writing quality assertions: manual/ language-based, template-based, and AI-based [3]. The manual way of writing assertions is confined to specific languages such as SystemVerilog Assertions (SVA) [4] and Property Specification Language (PSL) [6]. These approaches are very time-consuming and have become even more challenging when dealing with complex temporal logic from the specification. On the other hand, template-based approaches allow us to write assertions for multiple languages, such as OVL (Open Verification Language) [7]. However, these are limited to a specific scenario and can look at critical corner cases in the design.

AI-based methods, particularly those using Natural Language Processing (NLP) and LLMs, are prominent in this automation. As assertion generation is an NLP task, it has been an active area of research. For example, [8] proposed assertion generation using a dialogue between the NLP tool and the designer. On the other hand, the work of [9] involved latent representation of the language and FSM representation to extract the assertions out of it. Hybrid approaches combine rule-based systems with machine learning, as demonstrated in [10], which integrates spicy rules with GPT-3 to convert specification sentences into SVA language. Similarly, the research by [11] combined information retrieval and deep learning to generate assertions. These NLP methods offer automation but require extensive domain-specific data and training.

The study by [12] has discussed the applicability of LLMs in design, verification, and optimization; it describes how LLMs have shown transformative potential in EDA processes. For example, the work of [13] focuses on domain-adapted LLMs for assertion generation. However, this required data for training and adapting the existing foundation models for our task.

The approaches of [8] [10] focus on pre-RTL design for assertion generation, aiming for functional verification; on the other hand, [14] shows on RTL to generate assertions, but here it aims to verify the security assertion of that RTL. Very little work is involved in generating assertions from the specification documents using LLM. For example, more recently, the investigation conducted by [15] has generated assertions using multiple LLMs while solely on HLS documents without involving designer intent. Moreover, for example, the analysis presented in [16] showed RAG's application to reduce hallucinations in structured outputs. Similarly, The study of [17] introduces a multi-level specification framework to address understanding gaps between architects and designers.

These diverse methodologies underscore the evolving landscape of automated assertion generation. However, many of these approaches overlook the designer's intent or struggle with hallucinations in generated assertions. This gap in the literature sets the stage for our proposed framework using RAG and LLMs (given in Section III) to enhance VLSI design verification by bridging the understanding gap and improving the accuracy of generated assertions.

## III. Proposed Approach

### A. Abstract Workflow

To overcome the gap in the literature, we propose RAG in combination with LLMs for better quality assertion generation consistent with the HLS context. Fig. 2 illustrates the pipeline flow of our proposed RAG-LLM framework for assertion generation. Our proposed approach can be divided into two broader categories: the RAG part and the LLM part. For RAG, we aim to store the semantic context of HLS documents and possibly other relevant documents that could be useful in the synthesizing and decomposition processes in the VLSI design flow. On the other hand, the LLM part involves using actual specifications tailored by the designer or verification engineer that we intend to translate into assertion. The quality is evaluated by using the FPV technique against the golden design. The details of these parts are given in the following Sections.

### B. RAG-Part: HLS

From Fig.2, RAG-Part starts by taking HLS documents from the architect's side and given to the OpenAI chunker (Tiktoken) to fit the context window of the LLM easily, as shown in Step 1. In Steps 2 and 3, LLM-1 and LLM-2 do the processing over the chunk, respectively. LLM-1 acts as a Spec Analyzer that extracts useful specs-related information from the chunks and passes extracted specs to LLM-2. That acts as a Signal Mapper to make extracted spec consistency

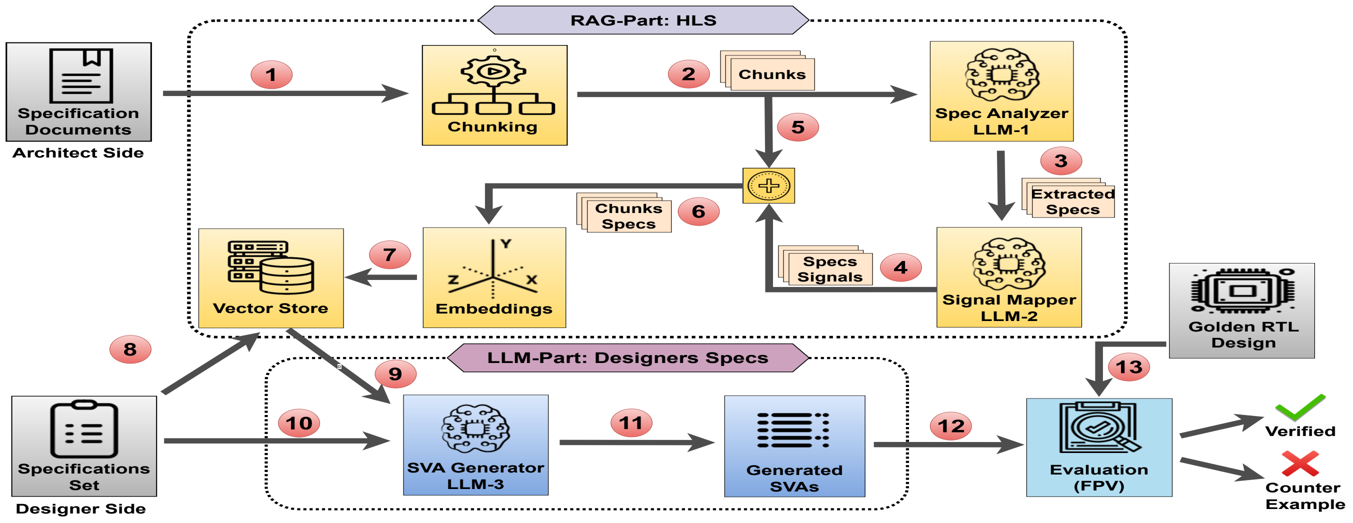

Fig. 2. Proposed Approach: RAG-LLM Assertion Generation

with HDL keywords and gives output Specs-Signals against chunks (shown in Steps 3 and 4). These steps make chunks more representable and better searchable for retrieval involving keywords. After that, In Steps 5 and 6, Specs-Signals are attached to their relevant chunk and passed as Chunks-Spec for embedding (semantic representation). We implement OpenAI embedding keys because they are comparable to our LLMs (GPT3.5 from OpenAI). Step 7 stores the HLS embeddings in an open-source ChromaDB vector database. The whole RAG process is implemented using an open-source LangChain framework. This RAG part has become an active source of knowledge to contextualize the upcoming assertion generation over designer-side specifications given in LLM-Part.

### C. LLM-Part: Designers Specs

In LLM-Part, designer-tailored specifications are input; the first is embedded and sent to ChromaDB (Vector Store) as a query. The query returns the semantically similar HLS context, illustrated in Steps 8 and 9. Then, in Step 10, our actual design side specification, which we intended to generate assertions (SVA), is given to our main LLM-3, the SVA generator. This LLM-3, with a custom prompt, is now more potent because the specification is to translate into SVA, in addition to its context from HLS; in the end, Step 11 generates SVA consistent with its HLS. Further, in the next part, the quality of the generated assertion can be evaluated against the golden RTL design.

### D. Evaluation-Part: FPV

In evaluation Steps 12 and 13, the quality of the generated assertions is checked by applying the formal property verification (FPV) technique using a bounded Model Checker (BMC). The generated assertions are checked against the golden RTL design via code instrumentation. If the specification is consistent with HLS documents, no counterexample is found. The results show (given in Section IV) the effectiveness of our approach. For implementing FPV, we use the Tabby CAD tool powered by YosysHQ that provides a set of tools, including an open-source software suite [18]. One of the main reasons for choosing this tool was the multiple choice of tool selection and usage options such as proof engine and task selection. Most tools are open-source. We applied our approach to open-source AMBA, AXI4-Lite bus communication protocols; this is the sub-protocol of a more extensive class of communication protocols Ax3, AXI4, AXI5, and all of their specifications are available by ARM [19].

Further detail on implementation tools and technologies is given in Section IV-A

## IV. Experimental Results

### A. Experimental Setup

The HLS document AXI4-Lite protocol [19] was in PDF; we first translated it into markdown(.md) format along with table and figures by applying state-of-the-art OCR techniques with LLM support. By doing this, we preserve the nature of the HLS document. On the other hand, The designer-tailored specifications were in a .txt file. Then, the whole RAG setup was implemented using the Langchain framework [20]. Table I summarizes the tools and technologies used to implement the two leading technologies, RAG and LLM, merged to generation design/ domain-specific quality assertion (SVA). We used pre-trained language models to generate SystemVerilog Assertions (SVA) for an open-source case study, with most tools also being open-source. Our central methodology uses a relatively economical Chat-completion API of the GPT 3.5 pre-trained model; with RAG, it becomes powerful. We chose the GPT model because these foundation models are state-of-the-art on most benchmarks. The setup and experiments were performed on Google Collab with T4 GPU support.

This experiment uses custom prompt engineering methods such as chain-of-thoughts and zero-shot learning to guide the LLMs to act as roles dedicated to them and get output in specific template formats. Using prompting, we tune LLM-1 (Spec Analyzer), LLM-2 (Signal Mapper), and LLM-3 (SVA-Generator). Our custom prompts are similar to the given in the

| NLP Approaches | NLP Tools/ Technologies | NLP Tasks |
|---|---|---|
| **Pre-processing** | Google OCR with GenAI, NLTK with Sklearn | Text extraction, text pre-processing |
| **RAG-implementation** | Tiktoken, OpenAI Embeddings, ChromaDV (LangChain) | Chunking, embeddings, storing in vector database, and querying |
| **LLM-implementation** | OpenAI GPT3.5, Chain-of-thought Fine-tuning | Pre-trained language models, prompt engineering |

TABLE I
SUMMARY OF EXPERIMENTAL SETUP

paper (using role and template-strategy) [15]. We added some extra constraints in the output template for getting assertions in concurrent SVA form to verify the formal temporal properties/ design intents of the signals. As LLM is inherently based on probability distribution in transformer architecture, it could deviate from its defined prompt template/ format, more likely when it starts maintaining the history. We performed our experiments with prompts five times to minimize this risk in a real scenario. We selected the results based on a 3-time occurrence at least, especially in case of the wrong examples. For example, to check the ability of our approach, we have given some wrong examples from designer-side specifications that were inconsistent with HLS. Some evaluation criteria are defined in Section IV-B.

## B. Evaluation Criteria

The literature commonly uses statistical evaluation metrics like Bleu and Meteor in NLP tasks. However, these metrics are not worth measuring hallucinations or the quality of generated assertions using LLM. Because statistical metrics rely on n-gram string comparison, it could be possible that different styles of assertions may have identical specifications. Moreover, no concrete benchmark is available for comparing the conformance to HLS (possible future research). Furthermore, this study does not deal with the coverage metrics, like calculating the cone of influence (COI) in SVA, because they go to the dynamic analysis side [21]. So, we have explored two metrics to measure the hallucination generated by conformance to HLS: human evaluation and applying formal verification such as FPV. In the following, we design evaluation metrics for measuring the tasks, such as hallucination-reduction and HLS-Conformity.

*1) Evaluation Metrics:*

- **Visual Evaluation (VE):** The hallucinated assertions include extra symbols or assumptions inconsistent with the HLS. The visual evaluation (VE) metric involves visually inspecting the assertions by a VLSI designer or verification engineer. Irrelevant symbols from hallucinations are highlighted in red, while information consistent with the HLS is green. This metric provides insight into the quality of the generated assertions.
- **Bounded model checker (BMC):** This tool checks the correctness of the generated assertions against the golden RTL design. The verified status shows that the generated assertion has been satisfied by the RTL design (colored green). A counter-example (colored red) indicates that the golden design does not meet the generated assertion/ property. A fail response highlights that the generated assertion is not executable (colored red). This metric is pivotal in building confidence in the LLM-generated assertions.

Under the given evaluation metrics, we can summarize our evaluation tasks as below:

*2) Evaluation Tasks:*

- **Hallucination-Reduction:** To check the quality of the generated assertion, hallucination is our mean; when the model gives un-useful responses, they can mislead our ABV process. Both evaluation metrics, VE and BMC, will be applied to measure the effectiveness of hallucination reduction using our RAG-LLM approach.
- **HLS-Conformity:** Checking the Conformance of the generated assertion to the context of the HLS document. Here, we also apply both evaluation metrics, VE and BMC, to evaluate the approach's effectiveness.

## C. Evaluation Results

To assess the effectiveness of the proposed RAG-LLM approach for assertion generation, we applied it to the AXI4-Lite protocol case study. We collected and prepared open-source HLS documents, designer-side specifications, and the golden RTL design from [19], [22]. Extensive experiments have been conducted, with ongoing development and evaluation. The details can be found on our GitHub repository. The following sections present the evaluation results using defined evaluation metrics as proof of concept.

*1) Hallucination-Reduction Results:* For establishing the hallucination reduction results. We compare the evaluation results of the proposed approach to GPT 3.5 and GPT 4 models, as examples shown in Table II. Using evaluation metrics (VE and BMC), the table demonstrates that our method generates more context-aware and accurate assertions (green) compared to GPT-3.5 and GPT-4. For example, GPT-3.5 produced assumptions (hallucinations) marked in red, assigning random bits to address signals and failing BMC with counterexamples (CE). While GPT-4 performed better. It still generated some misleading assertions. In specification 1, our approach used !$isunknown to handle signals accurately, whereas GPT-4 missed this, leading to potential false positives in formal BMC checks.

*2) HLS-Conformity Results:* Similarly, we conduct experiments on sample specification to evaluate the conformance of the generated assertions to the HLS. Table III shows seven sample specifications translated into SVA. The first five specifications were successfully translated, with the HLS-Conformance Excerpt column (green) indicating alignment with the high-level design specification achieved through RAG and LLMs. We used the FPV approach with BMC to verify these assertions against the golden RTL design, with no counterexamples confirming the specifications. We applied two

| No. | Spec | GPT 3.5 | GPT 4 | Our Approach | Evaluation Criteria |
|---|---|---|---|---|---|
| 1 | Ensure AWVALID is never asserted without AWADDR being valid | property p1; (posedge clk) disable iff (rst_n === 0) (AWVALID == 1’b1) \|->(AWADDR == 1’b1); endproperty assert property (p1); | property p1;(posedge clk) disable iff (reset) (AWVALID && !AWADDR_valid) \|->0; endproperty assert property (p1) else $error(”Assertion failed: AWVALID is asserted without AWADDR being valid”); | property p1; (posedge clk) disable iff (!rstn) (AWVALID \|->!$isunknown(AWADDR)); endproperty assert property (p1); | Visual Evaluation-VE (Human) |
| | | Assumptions + Failed | Misleading CEs | Verified | BMC |
| 2 | Ensure ARVALID is never asserted without ARADDR being valid | property p2; (posedge clk) disable iff (rst_n === 0) (ARVALID == 1’b1) \|->(ARADDR == 1’b1);endproperty assert property (p2); | property p2; (posedge clk) disable iff (reset) ARVALID \|->ARADDR_valid; endproperty assert property (p2) else error(”Assertion failed: ARVALID is asserted without ARADDR being valid”); | Property p2; (posedge clk) disable iff (!rstn) (!ARVALID \|->!$isunknown(ARADDR)); endproperty assert property (p2) | Visual Evaluation-VE (Human) |
| | | Hallucination + Failed | Misleading CEs | Verified | BMC |

TABLE II
HALLUCINATION-REDUCTION :SUMMARY OF THE TWO SPECS RESULTS

| No. | Spec | HLS-Conformance Excerpt | SVA | FPV(BMC) |
|---|---|---|---|---|
| 1 | Ensure AWVALID is never asserted without AWADDR being valid | Signals AWVALID and AWADDR which are used in the AXI write address channel. The context also explains that these signals work together to ensure proper write address information is transmitted. This aligns with the requirement in the specification sentence to ensure AWVALID is never asserted without AWADDR being valid. | property p1;(posedge clk) disable iff (!rstn)(AWVALID \|->!$isunknown(AWADDR)); endproperty assert property (p1); | Verified |
| 2 | Ensure ARVALID is never asserted without ARADDR being valid | - Conforming: The given SystemVerilog assertion is consistent with the specification since it ensures RREADY is only asserted when RVALID is high by using the global clock signal ’clk’ and global reset signal ’rstn’. | Property p2; (posedge clk) disable iff (!rstn) (!ARVALID \|->!$isunknown(ARADDR)); endproperty assert property (p2) else $error(”ARVALID asserted without valid ARADDR”); | Verified |
| 3 | Ensure WVALID is never asserted without WDATA being valid | - Signal WVALID fulfills the condition specified in the assertion as it is never asserted without WDATA being valid. | Property p3; (posedge clk) disable iff (!rstn) WVALID \|->WVALID && !$isunknown( WDATA);endproperty assert property (p3); | Verified |
| 4 | Ensure RREADY is asserted only when RVALID is high | The given SystemVerilog assertion is consistent with the specification since it ensures RREADY is only asserted when RVALID is high by using the global clock signal ’clk’ and global reset signal ’rstn’. | property p4; (posedge clk) disable iff (!rstn) (RREADY \|->RVALID)); endproperty assert property (p4) else $error(”RREADY is asserted when RVALID is low!”); | Verified |
| 5 | When AWVALID and AWREADY are high at the same time, the next cycle AWVALID goes low | Conforming: In both Spec 1 and Spec 2, AWVALID is defined as Master Write address valid and AWREADY is defined as Slave Write address ready, which confirms to the requirement in the spec. | property p5; (posedge clk) disable iff (!rstn) // Global clock and reset AWVALID && AWREADY && $rose(clk) \|->##1 !AWVALID;endproperty assert property (p5); | Verified |
| 6 | Ensure BREADY is asserted only when BVALID is high | This context does not provide specific information related to BREADY and BVALID signals. | property p6; logic BVALID; logic BREADY; (posedge clk) disable iff (!rstn) (BREADY \|->##1 BVALID); endproperty assert property (p6) else $error(B̈READY is asserted when BVALID is not higḧ); | Counter Example (CE) |
| 7 | Ensure RREADY is asserted only when BVALID is high | The provided specification context does not contain direct information related to RREADY and BVALID signals to confirm conformity with the generated SVA. | property p7; @(posedge clk) disable iff (!rstn) ( .AWVALID && !AWREADY) \|->!WREADY; endproperty assert property (p7) else $error(”Violation: RREADY is asserted when BVALID is low”); | Counter Example (CE) |

TABLE III
HLS-CONFORMITY: SUMMARY OF THE RESULTS (RAG-LLM)

incorrect specifications (specs 6 and 7) for further validation. These did not conform to the HLS, indicated by the red color, and generated assertions that failed with counter-examples in every case, demonstrating our approach’s effectiveness in correct and incorrect scenarios.

## V. DISCUSSION

This research uses an RAG implementation approach for a high-level specification that could perform well with advanced language models such as GPTs and Llama3. A small language/ foundation model may not be practical or expressive because of its limited ability to extract text. There is uncertainty in the models; we tried to confine the model output using the prompting technique, but LLM could be wired in its behavior. If LLM models start learning, the uncertainty could increase. There needs to be more certainty about the data on which language models were trained.

This technique’s Broader benefit can be its use in subsequent design processes in VLSI other than assertion generation. For example, this technique could be used for RTL debugging, synthesizing RTL designs, or checking conformance. Also, this approach has a benefit over the traditional assertion generation, which uses VectorDB for querying through search engines and search based on semantically due to semantic representation of the knowledge as in embedding space. That could be easily accessible via queries and similarity search techniques such as cosine similarity search.

The approach helps identify inconsistency, incorrectness, and incompleteness in the specification itself because it could be compared with all related design documents using our proposed methodology concept of RAG. It is also a cost-effective solution because our approach used GPT 3.5 API with RAG and custom prompt, outperforming the other techniques on zero short prompting. In the future, we can shift this setup to open-source LLMs such as the Llama4 model and get valuable insights and comparisons.

The proposed approach can also be integrated with existing EDA tools and VLSI workflow because of its nature of language prompting ability and adjustability against the prompt. There is no involvement in the preparation of the training dataset. It is a self-interpretative technique for generated assertions. When the specification standard changes, the designer can check the conformance of generated assertions under the latest specification documents. This technique gives the designer more freedom to enter the design implementation and verification. Designers can provide immediate feedback and refine specifications, leading to a continuous improvement loop. This loop enhances the design and the verification process by focusing on specific design intents concerning the designer.

## VI. Conclusion & Future Work

In this research, we proposed an innovative assertion generation framework from design specification by leveraging the RAG, a crucial NLP technique. The high-level specification (HLS) document(s) are embedded and stored in vectorDB as contextual knowledge to support the framework. This knowledge-based approach was an overall reference in conformance with the other VLSI design stages. Reducing LLM hallucination regarding design specification, RAG provides a knowledge source while generating assertions. To evaluate our approach, we implemented the proposed approach using FPV with a bounded model checker to check the functional correctness of the generated assertions against the gold RTL design. We implemented our approach to the AXI4-Lite bus communication protocol. We evaluated the conformance of the generated assertions to the HLS. We applied some wrong examples, which were also inconsistent with the HLS; in that case, the FPV checker showed a counter-example. In conclusion, this research is a step toward improving reliability and conformity to the HLS documents.

Future Work: We can first define or prepare benchmarks for better evaluation and implement them in the RTL design generation process. Secondly, in the future, we will implement this approach on more extensive industrial case studies from the automotive industry. Checking conformity with safety standards would be an exciting direction. Lastly, we plan to perform a detailed analysis of SVAs of multi-cycle, conditional, and sequential constraints, which could be very useful in exploring the corner cases of the designs and improving our verification.

## Ethical Consideration

Our approach is to reduce the hallucination and improve conformance with high-level specifications; we don't claim the risk of harm due to hallucination eliminated.

## References

[1] Saurabh, S. (2023). Introduction to VLSI design flow. Cambridge University Press.

[2] Dasgupta, P., & DasGupta, P. (2006). A roadmap for formal property verification (pp. 217-241). Springer Netherlands.

[3] Witharana, H., Lyu, Y., Charles, S., & Mishra, P. (2022). A survey on assertion-based hardware verification. ACM Computing Surveys (CSUR), 54(11s), 1-33.

[4] Mehta, A. B. (2014). SystemVerilog assertions and functional coverage. In Springer eBooks. https://doi.org/10.1007/978-1-4614-7324-4

[5] Rawte, V., Sheth, A., & Das, A. (2023). A survey of hallucination in large foundation models. arXiv preprint arXiv:2309.05922.

[6] Eisner, C., & Fisman, D. (2007). A practical introduction to PSL. Springer Science & Business Media.

[7] 2024. Open Verification Language. https://www.accellera.org/downloads/standards/ovl.Accessed:2024-07-01.

[8] Keszocze, O., & Harris, I. G. (2019, September). Chatbot-based assertion generation from natural language specifications. In 2019 Forum for Specification and Design Languages (FDL)(pp. 1-6). IEEE.

[9] Frederiksen, S. J., Aromando, J., & Hsiao, M. S. (2020, November). Automated assertion generation from natural language specifications. In 2020 IEEE International Test Conference (ITC)(pp. 1-5). IEEE.

[10] Aditi, F., & Hsiao, M. S. (2022, November). Hybrid rule-based and machine learning system for assertion generation from natural language specifications. In 2022 IEEE 31st Asian Test Symposium (ATS) (pp. 126-131). IEEE.

[11] Yu, H., Lou, Y., Sun, K., Ran, D., Xie, T., Hao, D., ... & Wang, Q. (2022, May). Automated assertion generation via information retrieval and its integration with deep learning. In Proceedings of the 44th International Conference on Software Engineering (pp. 163-174).

[12] He, Z., & Yu, B. (2024, March). Large Language Models for EDA: Future or Mirage?. In Proceedings of the 2024 International Symposium on Physical Design (pp. 65-66)

[13] Liu, M., Kang, M., Hamad, G. B., Suhaib, S., & Ren, H. (2024, April). Domain-Adapted LLMs for VLSI Design and Verification: A Case Study on Formal Verification. In 2024 IEEE 42nd VLSI Test Symposium (VTS) (pp. 1-4). IEEE.

[14] Kande, R., Pearce, H., Tan, B., Dolan-Gavitt, B., Thakur, S., Karri, R., & Rajendran, J. (2023). Llm-assisted generation of hardware assertions. arXiv preprint arXiv:2306.14027.

[15] Fang, W., Li, M., Li, M., Yan, Z., Liu, S., Zhang, H., & Xie, Z. (2024). Assertllm: Generating and evaluating hardware verification assertions from design specifications via multi-llms. arXiv preprint arXiv:2402.00386.

[16] Ayala, O., & Bechard, P. (2024, June). Reducing hallucination in structured outputs via Retrieval-Augmented Generation. In Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track) (pp. 228-238).

[17] Li, M., Fang, W., Zhang, Q., & Xie, Z. (2024). Specllm: Exploring generation and review of vlsi design specification with large language model. arXiv preprint arXiv:2401.13266.

[18] Tabby cad datasheet. About. (n.d.). https://www.yosyshq.com/tabby-cad-datasheet

[19] ARM. (2011). AMBA AXI and ACE Protocol specification. http://www.gstitt.ece.ufl.edu/courses/fall15/eel4720_5721/labs/refs/AXI4_specification.pdf

[20] Inc. , L. (n.d.). LangChain. LangChain. Retrieved June 20, 2024, from https://python.langchain.com/v0.1/docs/get_started/introduction

[21] Orenes-Vera, M., Martonosi, M., & Wentzlaff, D. (2023). Using llms to facilitate formal verification of rtl. arXiv e-prints, arXiv-2309.

[22] Vavintaparthi, B. R. (2023, December 21). Verification-of-AXI4-Lite-Bus-Protocol-using-System-Verilog-Assertions. Retrieved June 20, 2024, from https://github.com/balaji-vbr/Verification-of-AXI4-Lite-Bus-Protocol-using-System-Verilog-Assertions.