Review

Not peer-reviewed version

Hardware Design and Verification with Large Language Models: A Literature Survey, Challenges, and Open Issues

Meisam Abdollahi * , S. Faegheh Yeganli , Mohammad (Amir) Baharloo , Amirali Baniasadi

Posted Date: 4 November 2024

doi: 10.20944/preprints202411.0156.v1

Keywords: Large Language Model; Hardware Design; Hardware Verification; Hardware Accelerator; Debugging; Hardware Security; Hardware/Software Codesign

 doi:10.20944/preprints202411.0156.v1

Article

Hardware Design and Verification with Large Language Models: A Literature Survey, Challenges, and Open Issues

Meisam Abdollahi [1,*], S. Faegheh Yeganli [2], Mohammad (Amir) Baharloo [3], Amirali Baniasadi [1]

[1] Electrical and Computer Engineering Department, University of Victoria; amiralib@uvic.ca
[2] School of Engineering Science, Simon Fraser University; faegheh_yeganli@sfu.ca
[3] Electrical and Computer Science Department, University of Victoria; amirbaharloo@uvic.ca
* Correspondence: meisam@uvic.ca; Tel.: +1-(250)-661-2672(BC, Canada)

Abstract: Large Language Models (LLMs) are emerging as promising tools in hardware design and verification, with recent advancements suggesting they could fundamentally reshape conventional practices. In this survey, we analyze over 54 research papers to assess the current role of LLMs in enhancing automation, optimization, and innovation within hardware design and verification workflows. Our review highlights LLM applications across synthesis, simulation, and formal verification, emphasizing their potential to streamline development processes while upholding high standards of accuracy and performance. We identify critical challenges, such as scalability, model interpretability, and the alignment of LLMs with domain-specific languages and methodologies. Furthermore, we discuss open issues, including the necessity for tailored model fine-tuning, integration with existing Electronic Design Automation (EDA)) tools, and effective handling of complex data structures typical of hardware projects. This survey not only consolidates existing knowledge but also outlines prospective research directions, underscoring the transformative role LLMs could play in the future of hardware design and verification.

Keywords: Large Language Model; Hardware Design; Hardware Verification; Hardware Accelerator; Debugging; Hardware Security; Hardware/Software Codesign

1. Introduction

1.1. Introduction to LLMs

Large Language Models (LLMs) such as OpenAI's GPT-4 [1], Google's Gemini [2], Google's Bidirectional Encoder Representations from Transformers (BERT) [3] and Denoising Autoencoder from Transformer (BART) [4] which is a transformer-based model introduced by Facebook, are at the forefront of Artificial Intelligence (AI) research, revolutionizing how machines understand and generate human language. These models process extensive datasets covering a wide spectrum of human discourse, enabling them to perform complex tasks including translation, summarization, conversation, and creative content generation [1–5]. Recent advancements in this field, driven by innovative model architectures, refined training methodologies, and expanded data processing capabilities, have significantly enhanced the ability of these models to deliver nuanced and contextually relevant outputs. This evolution reflects a growing sophistication in AI's approach to Natural Language Processing (NLP), positioning LLMs as crucial tools in both academic research and practical applications, transforming interactions between humans and machines [5–7].

[1] https://chatgpt.com
[2] https://gemini.google.com/app
[3] https://github.com/google-research/bert
[4] https://huggingface.co/docs/transformers/model_doc/bart

 doi:10.20944/preprints202411.0156.v1

This evolution can be traced back to early statistical language models like n-gram models, which simply predicted word sequences based on the frequencies of previous sequences observed in a dataset. Although these models provided a foundational approach for text prediction, their limited ability to perceive broader contextual cues restricted their application to basic tasks [8–12]. The advent of neural network-based models, especially Recurrent Neural Networks (RNN), represented a significant advancement, offering the ability to retain information over longer text sequences and thus, managing more complex dialogues and text structures [13–15]. Although RNNs made advancements, they continued to struggle with scalability and long-term dependency issues, leading to the creation of Transformer models. These models introduced an innovative self-attention mechanism, allowing simultaneous processing of different sentence segments to enhance relevance and contextuality in text interpretation. This breakthrough underpins modern LLMs, which are pre-trained in extensive web-text data and subsequently fine-tuned for specific tasks, enabling them to generate nuanced, stylistically diverse, and seemingly authentic human text [16–21].

Moreover, with the advent of highly sophisticated models, LLMs have become an indispensable domain for both academic research and practical applications. These models necessitate thorough evaluations to fully understand their potential risks and impacts, both at task-specific and societal levels. In recent years, significant efforts have been invested in assessing LLMs from multiple perspectives, enhancing their applicability and effectiveness. The adaptability and deep comprehension abilities of LLMs have led to their extensive deployment across numerous AI domains. They are utilized not only in fundamental NLP tasks but also in complex scenarios involving autonomous agents and multimodal systems that integrate textual data with other data forms [22–39]. The utility of LLMs spans several domains, including healthcare [40–45], education [46–50], law [51–58], finance [59–63], and sciences [64–69], where they substantially improve data analysis and decision-making processes. This wide-ranging application underscores the transformative impact of LLMs on both technological innovation and societal functions.

In particular, within domains like hardware design and verification, LLMs enhance productivity and innovation by automating and optimizing various stages of the design process. These models can assist engineers in generating design specifications, suggesting improvements, and even creating initial design drafts. By leveraging vast amounts of data and advanced algorithms, LLMs can identify patterns and propose design optimizations that might not be immediately apparent to human designers. This capability helps in reducing time-to-market and ensuring that hardware designs are both efficient and innovative [70].

In hardware design, verification is a critical step in the lifecycle of hardware development. Verification ensures that the hardware performs as intended and meets all specified requirements before going into production. Traditionally, this process has been time-consuming and prone to human error. LLMs can automate much of the verification process by generating test cases, simulating hardware behavior, and identifying potential faults or discrepancies. They can analyze a large amount of verification data to predict potential issues and provide solutions, thus enhancing the reliability and accuracy of the hardware verification process. This not only speeds up the verification process, but also ensures a higher quality of the final product [71].

In addition to automation, LLMs facilitate better communication and collaboration among hardware design and verification teams. By providing a common platform where designers, engineers, and verification experts can interact with the model, LLMs help in bridging the gap between different teams. This collaborative approach ensures that all aspects of the hardware design and verification are aligned and that any issues are identified and addressed early in the process. Furthermore, LLMs can serve as knowledge repositories, offering solutions based on previous designs and verifications, thus ensuring that best practices are followed and past mistakes are not repeated [72].

Another significant benefit of LLMs in hardware design and verification is their ability to handle complex and high-dimensional data. Modern hardware designs are increasingly complex with numerous components and interdependencies. LLMs can manage this complexity by analyzing

 doi:10.20944/preprints202411.0156.v1

and processing large datasets to extract meaningful insights. They can model intricate relationships between different hardware components and predict how changes in one part of the design could impact the overall system. This holistic understanding is crucial for creating robust and reliable hardware systems [73].

In conclusion, integration of LLMs in hardware design and verification not only fosters innovation, but also ensures the development of cutting-edge hardware technologies [74,75]. This survey aims to explore the transformative role of LLMs in this domain, highlighting key contributions, addressing challenges, and discussing open issues that continue to shape this dynamic landscape. The goal is to provide a comprehensive overview that not only informs, but also inspires continued research and application of LLMs to improve hardware design and verification processes. We hope that this study will contribute to a better understanding and use of LLMs. In summary, the contributions of this paper can be summarized as follows:

- **Identification of Core Applications:** We detail the fundamental ways in which LLMs are currently applied in hardware design, debugging, and verification, providing a solid foundation to understand their impact.
- **Analysis of Challenges:** This paper presents a critical analysis of the inherent challenges in applying LLMs to hardware design, such as data scarcity, the need for specialized training, and integration with existing tools.
- **Future Directions and Open Issues:** We outline potential future applications of LLMs in hardware design and verification and discuss methodological improvements to bridge the identified gaps.

The remainder of the paper is organized as follows. Section 2 reviews the literature on the application of LLMs in hardware design and verification. Section 3 discusses the challenges associated with training and adapting LLMs for specific hardware design tasks. Section 4 explores open issues in the field and proposes areas for further investigation. Finally, Section 5 presents the conclusion of the study. The organization of the paper is depicted in Figure 1.

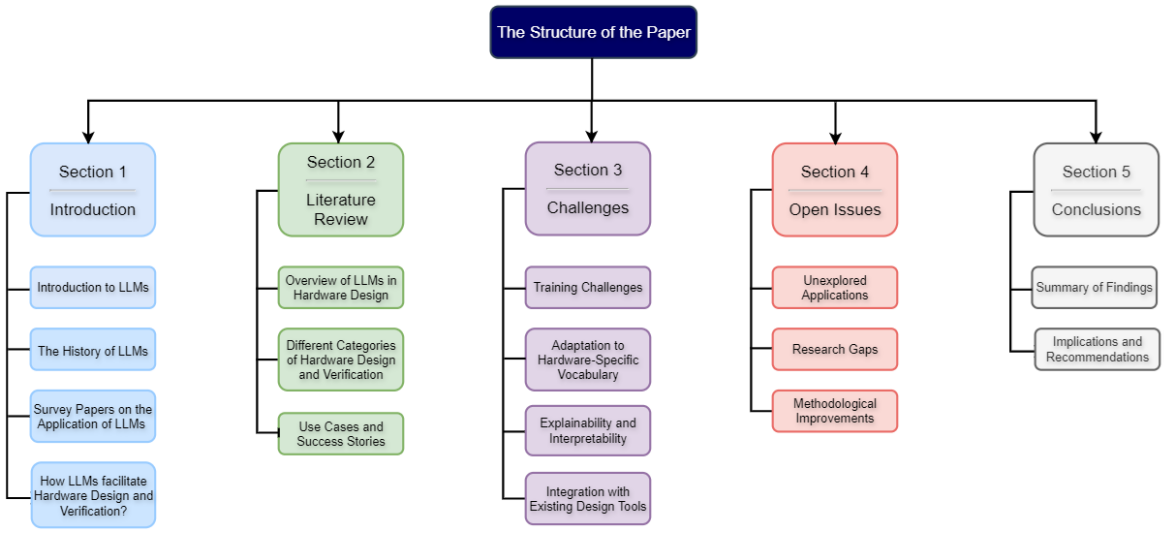

Figure 1. Section organization of the review paper

1.2. A Brief History of LLMs

The evolution of LLMs represents a crucial aspect of the broader development in AI. This progression begins with the earliest models and extends through to the sophisticated systems that today significantly influence computational linguistics and AI applications. A very brief history of LLMs is shown by Figure 2.

 doi:10.20944/preprints202411.0156.v1

Initially, LLMs operated on rule-based systems established in the mid-20th century, which were limited by strict linguistic rules and struggled to adapt to the variability of natural language. These systems, while foundational, offered limited utility for complex language tasks due to their inability to capture nuanced linguistic patterns [76]. The transition to statistical models like n-grams and Hidden Markov Models (HMMs) [77] in the late 20th century marked a pivotal enhancement. These models introduced a statistical approach to language processing, utilizing probabilities derived from large text corpora to predict language patterns. This shift allowed better handling of larger datasets, significantly improving real-world language processing capabilities. Despite these advancements, these models continued to struggle with deep contextual and semantic understanding, which later developments in algorithmic technology aimed to address [8–12].

By the early 2000s, the integration of advanced neural networks, specifically RNN [13–15] and Long Short-Term Memory (LSTM) networks [78], brought about substantial improvements in modeling sequential data. Additionally, the emergence of word embedding technologies like Word2Vec and GloVe advanced LLM capabilities by mapping words into dense vector spaces, capturing complex semantic and syntactic relationships more effectively. Despite these innovations, the increasing complexity of neural networks raised new challenges, particularly in model interpretability and the computational resources required [14,79,80].

The mid-2010s marked another significant advancement with the introduction of deep learning-based neural language models, notably the Recurrent Neural Network Language Model (RNNLM) in 2010, designed to effectively capture textual dependencies [81,82]. This development improved the generation of text that was more natural and contextually informed. However, these models also faced limitations such as restricted memory capacity and extensive training demands [83]. In 2015, Google's breakthrough with the Neural Machine Translation (GNMT) model utilized deep learning to significantly enhance machine translation, moving away from traditional rule-based and statistical techniques towards a more robust neural approach. This development not only improved translation accuracy but also addressed complex NLP challenges with greater efficacy [84,85].

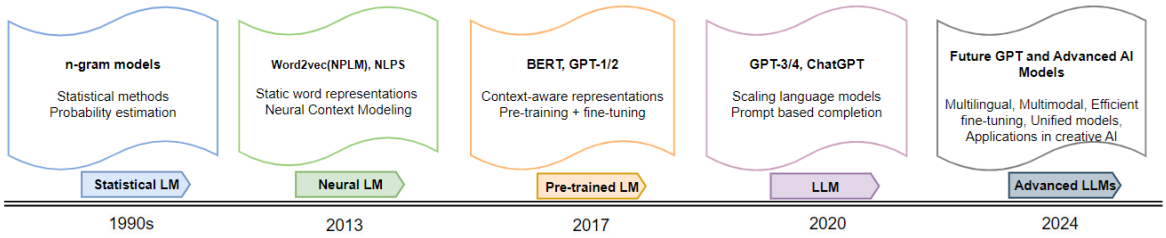

Figure 2. Brief history of language models [5].

A major breakthrough occurred in 2017 with the development of the Transformer model, which abandoned the sequential processing limitations of previous models in favor of self-attention mechanisms [16,17,86]. This innovation allowed for the parallel processing of words, drastically increasing efficiency and enhancing the model's ability to manage long-range dependencies. The Transformer architecture facilitated the creation of more sophisticated models such as BERT, which utilized bidirectional processing to achieve a deep understanding of text context, greatly improving performance across a multitude of NLP tasks [18,19,87]. Following BERT, models such as RoBERTa, T5, and DistilBERT have been tailored to meet the diverse requirements of various domains, illustrating the adaptability and expansiveness of LLM applications [88].

Subsequently, the introduction of OpenAI's GPT series further pushed the boundaries of what LLMs could achieve. Starting with GPT-1 and evolving through GPT-3, these models demonstrated exceptional capabilities in generating coherent and contextually relevant text across various applications. GPT-3, in particular, with its wide array of parameters, showcased the potential of LLMs to perform complex language tasks such as translation, question-answering and creative writing with minimal specific tuning. The advent of GPT-4 further broadened these capabilities by

 doi:10.20944/preprints202411.0156.v1

incorporating multimodal applications that process both text and images, thus significantly expanding the scope of LLMs. Recent developments, including enhancements in GPT-4 and the introduction of innovative models such as DALL-E 3, have continued this trend, emphasizing efficiency in fine-tuning, and enhancing capabilities in creative AI fields, demonstrating the versatility and depth of current models [19,21,87,89–92].

This progression from statistical models to today's advanced, multimodal, and domain-specific systems illustrates the dynamic and ongoing nature of LLMs development. These continual innovations not only advance the technology but also significantly impact the fields of AI and computational linguistics. Innovations such as sparse attention mechanisms, more efficient training algorithms, and the use of specialized hardware like Graphics Processing Units (GPUs) and Tensor Processing Units (TPUs) have enabled researchers to build increasingly larger and more powerful models. Moreover, efforts to improve model interpretability, reduce bias, and ensure ethical use are increasingly becoming central to the field.

In summary, the history of LLMs is a story of rapid progress driven by breakthroughs in Machine Learning (ML) and neural network architectures. From early statistical models to the transformative impact of the Transformer architecture and the rise of models like GPT-4, LLMs have evolved dramatically, reshaping our understanding of language and AI. As research continues, LLMs are poised to become even more integral to technological innovation and human-computer interaction in the years to come.

1.3. Survey Papers on the Application of LLMs in Different Areas

Due to the success of LLMs on various tasks and their increasing integration into AI research, examining the extensive survey literature on these models is essential. A significant number of surveys [5,93–103], provide detailed insights into the application of LLMs across different fields, demonstrating their advancements and wide-ranging uses. By analyzing these surveys, primarily published in 2022 and 2024, our aim is to gain a deeper understanding of LLMs' applications, and evaluate their potential impact across various sectors.

In this context, the work by Huang et al. [93] present a comprehensive survey of reasoning in LLMs, focusing on the current methodologies and techniques to enhance and evaluate reasoning capabilities in these models. The paper provides an in-depth review of various reasoning types, including deductive, inductive, and abductive reasoning, and discusses how these reasoning forms can be applied in LLMs. It also explores key methods used to elicit reasoning, such as fully supervised fine-tuning, prompting, and techniques like "chain of thought" prompting, which encourages models to generate reasoning steps explicitly. The authors review benchmarks and evaluation methods to assess reasoning abilities and analyze recent findings in this rapidly evolving field. Despite the advancements in reasoning with LLMs, the paper points out the limitations of current models, emphasizing that it remains unclear whether LLMs truly possess reasoning abilities or are merely following heuristics. Huang et al. conclude by offering insights into future directions, suggesting that better benchmarks and more robust reasoning techniques are needed to push the boundaries of LLMs' reasoning capabilities. This survey serves as an essential resource for researchers looking to understand the nuances of reasoning in LLMs and guide future research in this critical area. The authors review benchmarks and evaluation methods to assess reasoning abilities and analyze recent findings in this rapidly evolving field. Despite the advancements in reasoning with LLMs, the paper points out the limitations of current models, emphasizing that it remains unclear whether LLMs truly possess reasoning abilities or are merely following heuristics. Huang et al. conclude by offering insights into future directions, suggesting that better benchmarks and more robust reasoning techniques are needed to push the boundaries of LLMs' reasoning capabilities. This survey serves as an essential resource for researchers looking to understand the nuances of reasoning in LLMs and guide future research in this critical area. The authors review benchmarks and evaluation methods to assess reasoning abilities and analyze recent findings in this rapidly evolving field. Despite the advancements in reasoning with LLMs,

 doi:10.20944/preprints202411.0156.v1

the paper points out the limitations of current models, emphasizing that it remains unclear whether LLMs truly possess reasoning abilities or are merely following heuristics. Huang et al. conclude by offering insights into future directions, suggesting that better benchmarks and more robust reasoning techniques are needed to push the boundaries of LLMs' reasoning capabilities. This survey serves as an essential resource for researchers looking to understand the nuances of reasoning in LLMs and guide future research in this critical area. The authors review benchmarks and evaluation methods to assess reasoning abilities and analyze recent findings in this rapidly evolving field. Despite the advancements in reasoning with LLMs, the paper points out the limitations of current models, emphasizing that it remains unclear whether LLMs truly possess reasoning abilities or are merely following heuristics. Huang et al. conclude by offering insights into future directions, suggesting that better benchmarks and more robust reasoning techniques are needed to push the boundaries of LLMs' reasoning capabilities. This survey is an essential resource for researchers looking to understand the nuances of reasoning in LLMs and guide future research in this critical area.

In the study by Xi et al. [94], the authors comprehensively survey the rise and potential of LLM-based agents. The paper traces the evolution of AI agents, with a particular focus on LLMs as foundational components. It explores the conceptual framework of LLM-based agents, which consists of three main parts: brain, perception, and action. The survey discusses how LLMs, particularly transformer models, have been leveraged to enhance various agent capabilities, such as knowledge processing, reasoning, and decision-making, allowing them to interact effectively with their environments. Furthermore, the paper delves into the real-world applications of LLM-based agents across different sectors, including single-agent and multi-agent systems, as well as human-agent cooperation scenarios. It also highlights how LLM-based agents can exhibit behaviors akin to social phenomena when placed in societies of multiple agents. In addition, the study examines the ethical, security, and trustworthiness challenges posed by these agents, stressing the need for robust evaluation frameworks to ensure their responsible deployment. Finally, the authors present future research directions, particularly around scaling LLM-based agents, improving their capabilities in real-world settings, and addressing open problems related to their generalization and adaptability.

In [95], the authors present a detailed review of LLMs' evolution, applications, and challenges. The paper highlights the architecture and training methods of LLMs, particularly focusing on transformer-based models, and emphasizes their significant contributions across a range of sectors, including medicine, education, finance, and engineering. It also explores both the potential and limitations of LLMs, addressing ethical concerns such as biases, the need for vast computational resources, and issues of model interpretability. Furthermore, the survey delves into emerging trends, including efforts to improve model robustness and fairness, while anticipating future directions for research and development in the field. This comprehensive analysis serves as a valuable resource for researchers and practitioners, offering insights into the current state and future prospects of LLM technologies.

Naveed et al. [96] provide a comprehensive overview of LLMs, focusing on their architectural design, training methodologies, and diverse applications across various domains. The paper delves deeply into transformer models and their role in advancing NLP tasks. It also highlights the challenges associated with LLM deployment, including ethical concerns, computational resource demands, and the complexity of training these models. Additionally, the survey explores the impact of LLMs on different sectors such as healthcare, engineering, and social sciences, and identifies potential research directions for the future. This review serves as a key resource for researchers and practitioners looking to understand the current landscape of LLM development and deployment.

Fan et al. [97] present a comprehensive bibliometric analysis of over 5,000 publications on LLMs spanning from 2017 to 2023. This study aims to provide a detailed map of the progression and trends in LLM research, offering valuable insights for researchers, practitioners, and policymakers. The analysis delves into key developments in LLM algorithms and explores their applications across a wide range of fields, including NLP, medicine, engineering, and the social sciences. Additionally, the paper reveals

 doi:10.20944/preprints202411.0156.v1

the dynamic and fast-paced evolution of LLM research, highlighting the core algorithms that have driven advancements and examining how LLMs have been applied in diverse domains. By tracing these developments, the study underscores the substantial impact LLMs have had on both scientific research and technological innovation and provides a roadmap for future research in the field.

The study by Zhao et al. [5] offers an extensive survey of the evolution and impact of LLMs within AI and NLP. It traces the development from early statistical and neural language models to modern pre-trained language models (PLMs) with vast parameter sets. The paper highlights the unique capabilities that emerge as LLMs scale, such as in-context learning and instruction-following, which distinguish them from smaller models. A significant portion of the survey is dedicated to the contributions of LLMs, including their role in advancing AI applications like ChatGPT. Organized around four key areas—pre-training, adaptation tuning, utilization, and capacity evaluation—the study offers a comprehensive analysis of current evaluation techniques and benchmarks, while also identifying future research directions for enhancing LLMs and exploring their full potential.

In the study by Raiaan et al. [98], the authors conduct a comprehensive review of LLMs, focusing on their architecture, particularly transformer-based models, and their role in advancing NLP tasks such as text generation, translation, and question answering. The paper explores the historical development of LLMs, beginning with early neural network-based models, and examines the evolution of architectures like transformers, which have significantly enhanced the capabilities of LLMs. It discusses key aspects such as training methods, datasets, and the implementation of LLMs across various domains including healthcare, education, and business.

In another study, Minaee et al. [99] survey on LLMs illustrates an insightful analysis of the rise and development of LLMs, focusing on key models like GPT, LLaMA, and PaLM. The paper offers a comprehensive analysis of their architectures, training methodologies, and the scaling laws that underpin their performance in natural language tasks. Additionally, the survey examines key advancements in LLM development techniques, evaluates commonly used training datasets, and compares the effectiveness of different models through benchmark testing. Importantly, the study explores the emergent abilities of LLMs—such as in-context learning and multi-step reasoning—that differentiate them from smaller models, while also addressing real-world applications, current limitations, and potential future research directions.

In [100], the authors provide an in-depth analysis of the methodologies and technological advancements in the training and inference phases of LLMs. The paper explores various aspects of LLM development, including data preprocessing, model architecture, pre-training tasks, and fine-tuning strategies. Additionally, it covers the deployment of LLMs, with a particular emphasis on cost-efficient training, model compression, and the optimization of computational resources. The review concludes by discussing future trends and potential developments in LLM technology, making it a valuable resource for understanding the current and future landscape of LLM research and deployment.

Cui et al. [101] present a comprehensive survey on the role of Multimodal Large Language Models (MLLMs) in advancing autonomous driving technologies. The paper systematically explores the evolution and integration of LLMs with vision foundation models, focusing on their potential to enhance perception, decision-making, and control in autonomous vehicles. It reviews current methodologies and real-world applications of MLLMs in the context of autonomous driving, including insights from the 1st WACV Workshop on Large Language and Vision Models for Autonomous Driving (LLVM-AD). The study highlights emerging research trends, key challenges, and innovative approaches to improving autonomous driving systems through MLLM technology, emphasizing the importance of multimodal learning for the future of autonomous vehicles. Additionally, it stresses the need for further research to address critical issues like safety, data processing, and real-time decision-making in the deployment of these models.

Chang et al. [102] thoroughly explores the essential practices for assessing the performance and applicability of LLMs. It systematically reviews evaluation methodologies focusing on what aspects of

 doi:10.20944/preprints202411.0156.v1

LLMs to evaluate, where these evaluations should occur, and the best practices on how to conduct them. The paper explores evaluations across various domains including NLP, reasoning, medical applications, ethics, and education, among others. It also highlights successful and unsuccessful case studies in LLM applications, providing a critical insight into future challenges that might arise in LLM evaluation and stressing the need for a discipline-specific approach to effectively support the ongoing development of these models.

Kachris [103] provides a comprehensive analysis of the various hardware solutions designed to optimize the performance and efficiency of LLMs. The paper explores a wide variety of accelerators, including GPUs, FPGAs, and Application-Specific Integrated Circuit (ASIC)s, providing a detailed discussion of their architectures, performance, and energy efficiency metrics. It focuses on the significant computational demands of LLMs, particularly in both training and inference, and evaluates how these accelerators help meet these demands. The survey also highlights the trade-offs in performance and energy consumption, making it a valuable resource for those seeking to optimize hardware solutions for LLM deployment in data centers and edge computing.

Table 1 provides a comparison between various review papers, categorizing them based on critical features such as LLM models, APIs, datasets, domain-specific LLMs, ML-based comparisons, taxonomies, architectures, performance, hardware specifications for testing and training, and configurations.

 doi:10.20944/preprints202411.0156.v1

Table 1. Comparison of LLM Review Papers

Paper	LLMs Model	LLMs API	LLMs Dataset	Domain LLMs	Taxonomy	LLMs Architecture	LLMs Configurations	ML Comparisons	Performance	Parameters and Hardware Specification	Scope	Key Findings	Methodology and Approach
Huang et al. [93]	✓	x	✓	x	✓	✓	✓	✓	✓	x	LLM reasoning abilities	Explores LLMs' reasoning abilities and evaluation methodologies	Reasoning-focused review
Xi et al. [94]	✓	x	✓	✓	x	✓	x	x	✓	x	LLM-based AI agents for multiple domains	Highlights potential for LLMs as general-purpose agents	Agent-centric analysis
Hadi et al. [95]	✓	x	✓	x	✓	✓	✓	x	✓	✓	Comprehensive review of LLMs, applications, and challenges	Highlights potential of LLMs in various domains, discusses challenges and limitations	Literature review and analysis
Naveed et al. [96]	✓	x	✓	x	✓	✓	✓	✓	✓	✓	Overview of LLM architectures and performance	Challenges and advancements in LLM training, architectural innovations, and emergent abilities	Comparative review of models and training methods
Fan et al. [97]	✓	x	✓	x	✓	✓	x	x	x	x	Bibliometric review of LLM research (2017-2023)	Tracks research trends, collaboration networks, and the evolution of LLM research	Bibliometric analysis using topic modeling and citation networks
Zhao et al. [5]	✓	✓	✓	x	✓	✓	✓	✓	✓	x	Comprehensive survey of LLM models, taxonomy	Detailed analysis of LLMs evolution, taxonomy, emergent abilities, adaptation, and evaluation	Thorough review, structured methodology and various benchmarks
Raiaan et al. [98]	✓	x	✓	✓	✓	✓	✓	✓	✓	✓	Comprehensive review of LLM architectures, applications, and challenges	Discusses LLM development, applications in various domains, and societal impact	Extensive literature review with comparisons and analysis of open issues
Minaee et al. [99]	✓	x	✓	x	✓	✓	✓	✓	✓	x	Comprehensive survey of LLM architectures, datasets, and performance	Comprehensive review of LLM architectures, datasets, and evaluations	Comprehensive survey and analysis
Liu et al. [100]	✓	x	✓	x	✓	✓	✓	x	✓	✓	Training and inference in LLMs	Cost-efficient training and inference techniques are crucial for LLM development	Comprehensive review of training techniques and inference optimizations
Cui et al. [101]	✓	x	✓	✓	✓	✓	✓	✓	✓	✓	MLLMs for autonomous driving with extensive dataset coverage	Explores the potential of MLLMs in autonomous vehicle systems	Survey focusing on perception, planning, and control
Chang et al. [102]	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	Comprehensive evaluation of LLMs across multiple domains and tasks	Details LLM evaluation protocols, benchmarks, and task categories	Survey of evaluation methods for LLMs
Kachris et al. [103]	✓	x	✓	✓	✓	✓	✓	✓	✓	✓	Hardware solutions for accelerating LLMs	Energy efficiency improvements through hardware	Survey on hardware accelerators for LLMs

 doi:10.20944/preprints202411.0156.v1

Table 2. Tasks in hardware design and verification which can be done by LLMs

Category	Task	Description
Design	HDL Code Generation	Automatically generate Verilog, VHDL, or SystemC code from high-level design descriptions or specifications.
	Design Specification Translation	Convert natural language specifications into formal design requirements or constraints.
	Design Optimization Suggestions	Provide recommendations for optimizing design parameters such as power, performance, and area.
	Component Selection	Suggest suitable components based on design requirements and existing libraries.
	Documentation Generation	Create detailed design documentation, including block diagrams, interface definitions, and data sheets.
	Design Space Exploration	Propose and evaluate different design alternatives based on specified criteria.
	IP Core Integration	Automate the integration of IP cores into larger systems, including interface matching and configuration.
Verification	Test Bench Generation	Automatically generate test benches, including stimulus and expected results, from high-level test plans.
	Test Case Generation	Create individual test cases based on design specifications and verification requirements.
	Bug Detection and Suggestion	Analyze simulation logs and error reports to identify potential bugs and suggest debugging steps.
	Assertion Generation	Generate assertions for formal verification to ensure the correctness of design behavior.
	Coverage Analysis	Analyze coverage reports to identify untested areas and suggest additional tests.
	Regression Test Management	Automate the organization, execution, and analysis of regression test suites.
	Simulation Script Generation	Create scripts for running simulations with different configurations and scenarios.
Collaborative and Supportive Tasks	Code Review Assistance	Provide automated feedback on HDL code quality, compliance with coding standards, and potential issues
	Documentation Summarization	Summarize lengthy documentation and highlight key points for quicker understanding.
	Training Material Creation	Generate tutorials, guides, and FAQs for training new team members on tools and processes.
	Knowledge Base Maintenance	Organize and maintain a knowledge base of best practices, common issues, and solutions.
	Natural Language Queries	Answer queries in natural language about design specifications, verification results, and other relevant topics.
Design and Verification Workflow Automation	Requirement Traceability	Track design requirements through all stages of development and verification, ensuring all requirements are met.
	Change Impact Analysis	Analyze the impact of design changes on the overall system and suggest necessary verification updates.
	Project Management Support	Assist in tracking project milestones, deadlines, and deliverables related to design and verification.
Advanced Automation	Design Validation	Validate design correctness against high-level specifications using formal methods and simulation.
	Error Diagnosis	Diagnose errors in simulation results and suggest possible fixes based on historical data.
	Performance Analysis	Perform detailed performance analysis and suggest improvements based on simulation data.
	Automated Synthesis:	Guide the synthesis process to optimize the design for specific targets (e.g., low power, high performance).

1.4. How LLM facilitate Hardware Design and Verification?

By automating repetitive tasks, providing intelligent suggestions, and facilitating better communication and documentation, LLMs significantly improve the efficiency and effectiveness of hardware design and verification processes. They enable engineers to focus on higher-level problem-solving and innovation, thereby accelerating the development cycle and improving the quality of hardware products. A comprehensive list of these tasks are listed on Table 2.

2. Literature Review

2.1. Overview of LLMs in Hardware Design

LLMs have become a transformative tool in the field of hardware design and verification, bringing significant advancements in efficiency and accuracy. These models, powered by sophisticated AI and NLP capabilities, can analyze and interpret vast amounts of documentation, code, and design specifications, which accelerate the initial phases of hardware design. Using LLMs, engineers can automate the generation of design documents, ensuring consistency and reducing human error. This automation not only speeds up the design process but also enables the exploration of more complex and innovative designs, as the model can provide insights and suggestions based on a wide array of previous designs and industry standards.

In the realm of hardware verification, LLMs play a crucial role in improving the robustness and reliability of hardware systems. Verification is a critical step that ensures that the designed hardware functions correctly under all specified conditions. LLMs can help generate comprehensive test cases,

 doi:10.20944/preprints202411.0156.v1

identifying potential edge cases that could be overlooked by human designers. In addition, they can analyze the results of these tests more efficiently, highlighting discrepancies and providing detailed diagnostics that can pinpoint the root causes of failures. This capability significantly reduces the time and resources required for verification, allowing quicker iterations and more reliable hardware products. As a result, the integration of LLMs into hardware design and verification workflows is increasingly essential to maintain a competitive advantage in the fast-paced tech industry.

These studies collectively illustrate the transformative potential of LLMs in hardware design and verification, offering new methodologies that enhance efficiency, accuracy, and innovation in the field. As technology continues to evolve, further research and development will likely uncover even more applications and benefits, solidifying the role of LLMs as a crucial tool in modern hardware engineering.

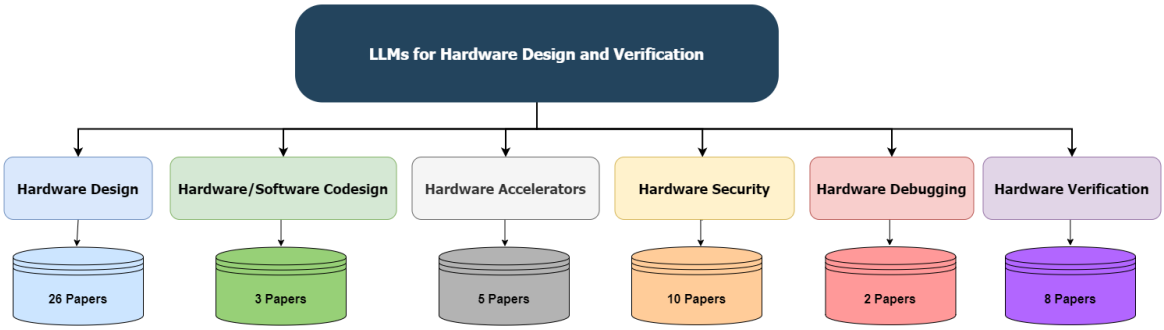

Figure 3. Different Categories of LLMs for Hardware Design and Verification

2.2. Different Categories of LLMs for Hardware Design and Verification

To the best of our knowledge, all articles in the literature can be categorized into six categories, as depicted in Figure 3. It should be noted that, although some papers could belong to two or more categories due to their approaches, we have decided to assign them to the category that most closely aligns with the majority of their content. Initially, we evaluated 62 articles. After the first round of review, 54 articles were selected for our survey. Hardware Design and Hardware Security were the most prominent categories, representing 48% and 18% of the articles, respectively. In the following sections, each category will be discussed in detail together with all surveyed papers.

> Create a Verilog module for a parameterized shift register with a default width of 8 bits. The shift register has clock (clk), reset (rst), enable, and scan enable control signals. On reset, it sets the internal data to 0. When enabled, it loads new input data (data_in), and when scan is enabled, it shifts in a single bit (scan_in). The internal state is output through data_out, and the most significant bit is provided as scan_out.

Figure 4. Input prompt for a HDL design

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 November 2024 doi:10.20944/preprints202411.0156.v1

The issue with your shift register occurs when `WIDTH` is set to 1, making the range `internal_data[WIDTH-2:0]` invalid because it results in a negative index. To fix this, you need to handle the case where `WIDTH` is 1 separately.

Here's the modified version of the code that works when `WIDTH` is 1:

verilog Copy code

```
module shift_register #(parameter WIDTH = 8) (
    // Ports omitted for space
);
    reg [WIDTH-1:0] internal_data;

    // Shift register operation
    always @(posedge clk) begin
        if (rst) begin
            internal_data <= {WIDTH{1'b0}};
        end else if (enable) begin
            internal_data <= data_in;
        end else if (scan_enable) begin
            if (WIDTH == 1) begin
                internal_data <= scan_in;
            end else begin
                internal_data <= {internal_data[WIDTH-2:0], scan_in};
            end
        end
    end

    // Output assignment
    assign data_out = internal_data;
    assign scan_out = internal_data[WIDTH-1];
endmodule
```

Key Changes:

- I added a conditional block to handle the case where `WIDTH == 1`. Instead of attempting to access a non-existent bit slice `internal_data[WIDTH-2:0]`, it simply assigns `scan_in` directly to `internal_data`.

With this change, the shift register will now work properly even when `WIDTH` is set to 1.

Figure 5. ChatGPT output for the input prompt of the HDL design

 doi:10.20944/preprints202411.0156.v1

Verilog

```
module shift_register #(parameter WIDTH = 8) (
    input clk, rst, enable, scan_enable, data_in, scan_in,
    output data_out, scan_out
);

    reg [WIDTH-1:0] internal_data;

    // Shift register operation
    always @(posedge clk) begin
        if (rst) begin
            internal_data <= {WIDTH{1'b0}};
        end else if (enable) begin
            internal_data <= data_in;
        end else if (scan_enable) begin
            if (WIDTH == 1) begin
                internal_data <= scan_in;
            end else begin
                internal_data <= {internal_data[WIDTH-2:0], scan_in};
            end
        end
    end

    // Output assignment
    assign data_out = internal_data;
    assign scan_out = internal_data[WIDTH-1];
endmodule
```

Use code with caution.

This Verilog module provides a parameterized shift register with the specified functionality. It incorporates the improvements from the previous responses, including a default width of 8 bits and a conditional check for WIDTH == 1 in the scan enable case.

Figure 6. Gemini output for the input prompt of the HDL design

 doi:10.20944/preprints202411.0156.v1

Figure 7. Copilot output for the input prompt of the HDL design

2.2.1. Hardware Design

Hardware design using HDLs is evolving with the integration of LLMs such as OpenAI's ChatGPT, Google Gemini, and Microsoft Copilot. These LLMs assist in modifying and generating HDL code by accepting design specifications as input prompts, streamlining the development process. In Figure 4, we demonstrate an HDL design for a shift register, where the input prompt describes the desired behavior and adjustments. The results of this input prompt processed by different LLMs are presented in Figures 5–7, showing the outputs of ChatGPT 4o, Gemini, and Copilot, respectively. These figures illustrate the varying approaches taken by each model to interpret and modify the HDL design.

This paper [104] presents a comprehensive framework for evaluating the performance of hardware setups used in the inference of LLMs. The framework aims to measure critical performance metrics such as latency, throughput, energy efficiency, and resource utilization to provide a detailed assessment of hardware capabilities. By standardizing these measurements, the framework facilitates consistent and comparable evaluations across different hardware platforms, enabling researchers and engineers to identify the most efficient configurations for LLM inference. The proposed framework addresses the growing complexity and computational demands of LLMs by offering a robust tool for hardware benchmarking. It integrates various testing scenarios to capture the diverse workloads LLMs handle during inference. This allows for a nuanced understanding of how different hardware components contribute to overall performance. Ultimately, the framework aims to guide the development of more

 doi:10.20944/preprints202411.0156.v1

efficient hardware solutions tailored to the specific needs of LLMs, promoting advancements in both hardware design and LLM deployment strategies.

This paper [105] investigates the application of LLMs in optimizing and designing VHDL (VHSIC Hardware Description Language (HDL)) code, a critical aspect of digital circuit design. The study explores how LLMs can automate the generation of efficient VHDL code, potentially reducing the time and effort required in the design process. Through a series of experiments, the authors demonstrate the capability of LLMs to provide high-quality code suggestions and optimizations, which can enhance the performance and reliability of digital circuits. The paper also discusses the challenges associated with integrating LLMs into the VHDL design workflow. It highlights issues such as the need for domain-specific training data and the importance of understanding the context and constraints of hardware design. By addressing these challenges, the study provides insights into how LLMs can be effectively utilized to support and streamline VHDL code development, paving the way for more automated and intelligent design processes in digital electronics.

AutoChip [106] introduces a novel method to automate the generation of HDL code by using feedback from LLMs. The approach involves an iterative process where LLMs provide suggestions and improvements on initial HDL code drafts, leading to refined and optimized final versions. This method significantly reduces the need for manual intervention, making the design process more efficient and accessible, especially for complex hardware projects. The paper outlines the technical details of implementing AutoChip, including training LLMs on domain-specific datasets and the integration of feedback loops into the design workflow. Case studies demonstrate the effectiveness of AutoChip in producing high-quality HDL code with minimal human oversight. By automating routine and complex coding tasks, AutoChip has the potential to revolutionize the field of hardware design, enabling faster prototyping and more innovative solutions in hardware development.

This study [107] focuses on benchmarking various LLMs to evaluate their performance in generating Verilog Register Transfer Level (RTL) code. The paper compares the accuracy, efficiency, and complexity handling capabilities of different models, providing a comprehensive assessment of their suitability for automated hardware design tasks. By establishing standardized benchmarks, the research offers valuable insight into the strengths and limitations of each model in the context of the generation of Verilog code RTL. The findings highlight significant differences in model performance, emphasizing the importance of selecting the right LLM for specific hardware design tasks. The paper also discusses the potential for further improving LLMs by incorporating more specialized training data and refining model architectures. Through detailed comparisons and practical examples, this study contributes to the ongoing effort to enhance the role of LLMs in automating and optimizing the hardware design process.

Chip-Chat [108] explores the emerging field of using conversational glsai, particularly LLMs, in hardware design. The paper discusses the potential benefits of natural language interfaces, such as increased accessibility and collaboration, by enabling designers to interact with design tools through simple conversational commands. This approach could democratize hardware design, making it more accessible to non-experts and fostering innovation through diverse contributions. However, the paper also highlights significant challenges in this domain, including the current limitations of LLMs in understanding complex hardware design concepts and the need for precise and unambiguous communication in technical contexts. The authors propose potential solutions, such as improving model training with domain-specific data and developing more sophisticated interaction protocols. By addressing these challenges, the paper aims to pave the way for more effective integration of conversational AI in hardware design, potentially transforming how engineers and designers approach complex projects.

ChipGPT [109] examines the current state of using LLMs for natural language-based hardware design, assessing their capabilities and identifying existing gaps. The paper evaluates various LLMs in terms of their ability to understand and generate hardware design code from natural language descriptions. It highlights the potential of these models to streamline the design process by enabling

 doi:10.20944/preprints202411.0156.v1

more intuitive and accessible interactions between designers and design tools. Despite the promising potential, the paper identifies several challenges that need to be addressed to achieve seamless natural language hardware design. These include improving the models' understanding of technical jargon and design constraints, enhancing the accuracy and efficiency of code generation, and ensuring robust handling of complex design scenarios. By outlining these challenges and suggesting areas for future research, the study provides a roadmap for advancing the integration of NLP in hardware design workflows.

This paper [110] explores the application of LLMs to detect code segments that can benefit from hardware acceleration. The study presents techniques for identifying computation-intensive parts of code that, when offloaded to specialized hardware, can significantly improve overall system performance. The authors demonstrate the effectiveness of LLMs in pinpointing these critical segments and providing recommendations for hardware acceleration. The research also discusses the practical implementation of this approach, discussing how LLMs can be integrated into existing development workflows to automatically suggest and optimize code for hardware acceleration. The paper highlights the potential performance gains and efficiency improvements achievable through this method, making a strong case for the use of LLMs to optimize software for better hardware utilization. By leveraging LLMs for this purpose, developers can achieve more efficient and powerful computing solutions.

CreativEval [111] introduces a novel approach to evaluating the creativity of hardware design code generated by LLMs. The paper proposes specific metrics and benchmarks to assess the originality, efficiency, and practicality of the generated code, emphasizing the importance of creativity in hardware design. By focusing on these aspects, the study aims to determine how well LLMs can innovate within the constraints of hardware development. The findings suggest that while LLMs are capable of producing creative solutions, there are limitations to their ability to fully replicate human ingenuity in hardware design. The paper discusses the potential for improving LLMs through more diverse and comprehensive training data, as well as refining the evaluation metrics to better capture the nuances of creative design. Through this evaluation framework, CreativEval contributes to the understanding of LLM capabilities in generating novel and effective hardware design solutions.

Designing Silicon Brains using LLM [112] explores the use of ChatGPT, a type of LLMs, for designing a spiking neuron array, which is a type of neuromorphic hardware that mimics the behavior of biological neurons. The authors demonstrate how ChatGPT can generate detailed and accurate descriptions of the neuron array, including its architecture and functionality, thereby aiding in the design process. This approach leverages the model's ability to understand and articulate complex technical concepts in natural language. The study showcases the potential of using LLMs for designing advanced neuromorphic systems, highlighting the benefits of automated description and specification generation. By providing a structured and comprehensive design output, ChatGPT can significantly streamline the development process of spiking neuron arrays. The paper also discusses the challenges and future directions for integrating LLMs into neuromorphic hardware design, emphasizing the need for further refinement and domain-specific training to enhance the accuracy and utility of the generated descriptions.

Digital ASIC Design with Ongoing LLMs [113] explores the application of ongoing LLMs in the design of ASICs. The paper provides an overview of current methodologies and strategies for incorporating LLMs into various stages of the ASIC design process, from initial specification to final implementation. It highlights the potential of LLMs to automate routine tasks, enhance design accuracy, and reduce development time. The authors also discuss the prospects and challenges of using LLMs in digital ASIC design, including the need for specialized training data, the integration of LLMs into existing design workflows, and the importance of maintaining design integrity and performance. By addressing these issues, the study offers valuable insights into the future of ASIC design, suggesting that LLMs could play a significant role in advancing the field through increased automation and intelligent design support.

 doi:10.20944/preprints202411.0156.v1

This paper [114] reviews recent advancements in the development of efficient algorithms and hardware architectures specifically tailored for NLP tasks. The authors discuss various techniques for optimizing NLP algorithms to improve performance and reduce computational requirements. These optimizations are crucial for handling large-scale data and complex models typical of modern NLP applications, including LLMs. The study also explores the design of specialized hardware that can support efficient NLP. This includes discussing hardware accelerators, such as GPUs and TPUs, and their role in enhancing the performance of NLP tasks. By combining algorithmic improvements with hardware advancements, the paper outlines a comprehensive approach to achieving high efficiency and scalability in NLP applications. This integrated approach is essential for meeting the growing demands of NLP and leveraging the full potential of LLMs.

GPT4AIGChip [115] investigates the use of LLMs to automate the design of AI accelerators. The paper presents methodologies for leveraging LLMs to generate design specifications and optimize the architecture of AI accelerators. By automating these processes, the study aims to streamline the development of specialized hardware for AI tasks, reducing both the time and costs associated with traditional design methods. The authors provide detailed case studies demonstrating the effectiveness of LLMs in producing high-quality design outputs for AI accelerators. They highlight the potential for LLMs to not only accelerate the design process but also to innovate and improve upon existing architectures. The paper discusses future directions for this research, including the integration of more advanced LLMs and the development of more sophisticated automation tools. By advancing the use of LLMs in hardware design, GPT4AIGChip contributes to the ongoing evolution of AI hardware development.

Hardware Phi-1.5B [116] introduces a LLM trained specifically on hardware-related data, demonstrating its capability to encode domain-specific knowledge. The paper discusses the model's architecture and training process, emphasizing the importance of specialized datasets for achieving high performance in hardware design tasks. The authors showcase various applications of Hardware Phi-1.5B, including code generation, optimization, and troubleshooting in hardware development. The study highlights the advantages of using domain-specific LLMs over general-purpose models, particularly in terms of accuracy and relevance of the generated content. By tailoring the model to the hardware domain, Hardware Phi-1.5B is able to provide more precise and contextually appropriate outputs, which can significantly enhance the efficiency and effectiveness of hardware design processes. The paper concludes with a discussion of future research directions and potential improvements to further leverage domain-specific LLMs in hardware engineering.

Hardware-Aware Transformers (HAT) [117] introduces a novel approach to designing LLMs that consider hardware constraints during model training and inference. The paper details the development of transformers that are optimized for specific hardware configurations, aiming to improve performance and reduce resource consumption. This hardware-aware design is particularly important for deploying NLP tasks on various platforms, including edge devices and specialized accelerators. The study presents experimental results demonstrating the efficiency gains achieved by HAT models compared to traditional transformers. The authors highlight significant reductions in latency and energy usage, making these models more suitable for real-world applications where resource constraints are a critical factor. By focusing on the co-design of hardware and software, HAT offers a promising solution for enhancing the performance and scalability of NLP tasks in diverse deployment environments.

This paper [118] proposes a post-processing technique to improve the quality of hardware design code generated by LLMs. The approach involves applying search algorithms to refine and optimize the initial outputs from LLMs, ensuring higher quality and more reliable code generation. The authors detail the implementation of this technique and provide experimental results demonstrating its effectiveness in enhancing code quality. The study highlights the limitations of current LLM-generated code, such as inaccuracies and inefficiencies, and shows how post-LLM search can address these issues. By iteratively refining the code, the proposed method can significantly improve the final output,

 doi:10.20944/preprints202411.0156.v1

making it more suitable for practical hardware design applications. This approach underscores the potential for combining LLM capabilities with additional optimization techniques to achieve superior results in automated code generation.

OliVe [119] introduces a quantization technique designed to accelerate LLMs by focusing on hardware-friendly implementations. The concept of outlier-victim pair quantization is presented as a method to reduce the computational load and improve inference speed on hardware platforms. This technique targets specific outliers in the data, which typically require more resources, and optimizes their representation to enhance overall model efficiency. The paper provides a detailed analysis of the quantization process and its impact on model performance. Experimental results demonstrate significant improvements in inference speed and resource utilization without compromising the accuracy of the LLMs. By making LLMs more hardware-friendly, OliVe offers a practical solution for deploying these models in environments with limited computational resources, such as mobile devices and edge computing platforms.

RTLCoder [120] presents a specialized model designed to outperform GPT-3.5 in generating RTL code. The paper highlights the use of an open-source dataset and a lightweight model architecture to achieve superior results in RTL code generation tasks. The authors provide a comprehensive comparison of RTLCoder's performance against GPT-3.5, demonstrating significant improvements in accuracy, efficiency, and code quality. The study also discusses the advantages of using a focused dataset and a streamlined model for specific applications, such as hardware design. By tailoring the model to the unique requirements of RTL code generation, RTLCoder can provide more relevant and high-quality outputs, reducing the need for extensive manual corrections. This approach underscores the potential for developing specialized models that can outperform general-purpose LLMs in targeted tasks.

RTLLM [121] introduces an open-source benchmark specifically designed to evaluate the performance of LLMs in generating RTL code. The benchmark provides a standardized set of tasks and metrics to facilitate consistent and comprehensive assessments of LLM capabilities in RTL generation. By offering a common evaluation framework, RTLLM aims to drive improvements and innovation in the use of LLMs for hardware design. The paper details the creation of the benchmark, including the selection of tasks, the development of evaluation criteria, and the compilation of relevant datasets. The authors present initial results from using RTLLM to evaluate various LLMs, highlighting strengths and areas for improvement. By providing an open-source tool for benchmarking, RTLLM encourages collaboration and transparency in the development and assessment of LLMs for RTL code generation, fostering advancements in this emerging field.

This paper [122] discusses the co-design approach to selecting features for language models, and balancing software and hardware requirements to achieve optimal performance. The authors highlight the importance of considering both aspects in the development of LLMs, as hardware constraints can significantly impact model efficiency and scalability. By integrating software and hardware design, the study aims to create more efficient and effective LLMs. The paper presents case studies and experimental results demonstrating the benefits of the co-design approach. These examples show how tailored features can enhance model performance on specific hardware platforms, reducing latency and resource consumption. The authors argue that this integrated approach is essential for developing LLMs that can meet the growing demands of real-world applications, offering a path forward for more sustainable and scalable NLP solutions.

SlowLLM [123] investigates the feasibility and performance of running LLMs on consumer-grade hardware. The paper addresses the challenges and limitations of deploying LLMs on less powerful devices, such as personal computers and smartphones, which often lack the computational resources of specialized hardware. The authors propose solutions to enhance performance, such as model compression and optimization techniques tailored for consumer hardware. The study provides experimental results showcasing the performance of various LLMs on consumer devices, highlighting both successes and areas for improvement. The findings demonstrate that, while there are significant

 doi:10.20944/preprints202411.0156.v1

challenges, it is possible to achieve acceptable performance levels with appropriate optimizations. By exploring these possibilities, SlowLLM contributes to making advanced NLP capabilities more accessible to a broader audience, potentially expanding the applications and impact of LLMs in everyday technology use.

SpecLLM [124] investigates the use of LLMs for generating and reviewing Very-Large Scale Integration (VLSI) design specifications. The paper evaluates the ability of LLMs to produce accurate and comprehensive specifications, which are crucial for the development of complex integrated circuits. The authors present methods for training LLMs on domain-specific data to enhance their understanding and performance in VLSI design tasks. The study provides experimental results demonstrating the effectiveness of LLMs in generating VLSI design specifications, highlighting their potential to streamline the design process and reduce errors. The authors discuss the challenges and future directions for improving the integration of LLMs in VLSI design, including the need for more sophisticated training techniques and better handling of technical jargon. By exploring these possibilities, SpecLLM contributes to the ongoing efforts to enhance the role of LLMs in the field of hardware design.

ZipLM [125] introduces a structured pruning technique designed to improve the inference efficiency of LLMs. The paper details the development of pruning methods that selectively remove less important components of the model, reducing its size and computational requirements without significantly impacting performance. This inference-aware approach ensures that the pruned models remain effective for their intended tasks while benefiting from enhanced efficiency. The study presents experimental results demonstrating the effectiveness of ZipLM in reducing model size and improving inference speed. The authors highlight the potential applications of this technique in environments with limited computational resources, such as edge devices and mobile platforms. By focusing on structured pruning, ZipLM offers a practical solution for deploying LLMs more efficiently, enabling broader accessibility and application of these powerful models in various real-world scenarios.

This paper [126] explores the application of advanced language models (ALMs), such as GPT-3.5 and GPT-4, in the realm of electronic hardware design, particularly focusing on Verilog programming. Verilog is a HDL used for designing and modeling digital systems. The study introduces the VeriPPA framework, which utilizes ALMs to generate and refine Verilog code. This framework incorporates a two-stage refinement process to enhance both the syntactic and functional correctness of the generated code and to align it with key performance metrics—Power, Performance, and Area (PPA). This iterative approach involves leveraging diagnostic feedback from simulators to identify and correct errors systematically, akin to human problem-solving techniques. The methodology begins with ALMs generating initial Verilog code, which is then refined through the VeriRectify process. This process uses error diagnostics from simulators to guide the correction of syntactic and functional issues, ensuring the generated code meets specific correctness criteria. Following this, the code undergoes a PPA optimization stage where its power consumption, performance, and area efficiency are evaluated and further refined if necessary. This dual-stage approach significantly improves the quality of Verilog code, achieving an 81.37% linguistic accuracy and a 62.0% operational efficacy in programming synthesis, surpassing existing techniques. The study highlights the potential of ALMs in automating and improving the hardware design process, making it more accessible and efficient for those with limited expertise in chip design.

 doi:10.20944/preprints202411.0156.v1

Table 3. Comparison of papers focuses on Hardware Design with LLMs

Parameter		Approach	References
Scope and Focus		Optimizing hardware specifically for LLM inference and performance	[104,115,117,127]
		Generation and optimization of hardware design code using LLMs	[105,106,113,124]
		Exploring broader challenges and opportunities in conversational and natural language-based hardware design	[108,109]
		Code detection for acceleration and quantization techniques	[110,119]
Methodologies	Benchmarking and Evaluation	Benchmarking and evaluating LLM performance in hardware-related tasks	[104,107,121,128]
	Automated Generation	Methodologies for automating HDL generation and design specification using LLM feedback	[106,120,124]
	Optimization Techniques	Exploring specific optimization techniques like hardware-aware transformers and structured pruning	[110,117,125,127]
Innovative Contributions	Creativity and Originality	Evaluating the creativity of LLM-generated hardware code	[111,128]
	Neuromorphic Hardware	Focusing on designing neuromorphic hardware (spiking neuron arrays) using LLMs, highlighting an innovative application in the field	[112]
	Consumer Hardware Feasibility	Investigating the feasibility of running LLMs on consumer-grade hardware, addressing practical deployment challenges	[123,127]
Application Areas	AI Accelerators	Automation of AI accelerator design, reflecting the growing importance of specialized hardware for AI tasks.	[115,120,127]
	VLSI Design	VLSI design specifications, an area critical for complex integrated circuit design.	[124]
	General Hardware Design	Looking at various aspects of hardware design and integration with LLMs.	[109,113,114,128]
Performance and Efficiency		Making LLMs more efficient and hardware-friendly, addressing the computational and resource challenges associated with large models.	[117,119,127]
		Discussing frameworks and techniques to enhance inference performance, which are crucial for deploying LLMs in real-world applications.	[104,125,128]

This paper [127] reviews and proposes various strategies to accelerate and optimize LLM, addressing the computational and memory challenges associated with their deployment. It covers algorithmic improvements, including early exiting and parallel decoding, and introduces hardware-specific optimizations through LLM-hardware co-design. The paper presents frameworks like Medusa for parallel decoding, achieving speedups up to 2.8x, and SnapKV for memory efficiency improvements. It also explores High Level Synthesis (HLS) applications with frameworks like ScaleHLS and HIDA, which convert LLM architectures into hardware accelerators. These advances improve LLM performance in real-time applications, such as NLP and EDA, while reducing energy consumption and improving efficiency.

From English to ASIC [128] explores the application of LLMs in automating hardware design using HDLs like Verilog for ASICs. The authors focus on improving the precision of LLM-generated HDL code by fine-tuning Mistral 7B and addressing challenges such as syntax errors and the scarcity of quality training datasets. By creating a labeled Verilog dataset and applying advanced optimization techniques such as LoRA and DeepSpeed ZeRO, the paper shows significant improvements in code generation accuracy, with up to a 20% increase in pass@1 metrics. The paper's contributions include optimizing memory usage and inference speed, making LLMs more practical for EDA in hardware development. Table 3 provides an overview of all the papers discussed in the hardware design subsection for comparison.

2.2.2. Hardware/Software Codesign

This paper [129] explores co-design strategies to improve training speed and scalability of deep learning recommendation models. The authors emphasize the integration of software and hardware design to achieve significant performance gains, addressing the computational intensity and resource demands of training large models. The study presents various techniques for optimizing algorithms and hardware architectures, ensuring efficient utilization of resources. The paper showcases experimental results demonstrating the effectiveness of co-design strategies in accelerating model training. These results highlight improvements in training times and scalability, making it feasible to handle larger datasets and more complex models. By focusing on the co-design approach, the study provides valuable insights into achieving faster and more scalable training processes, which are essential for the ongoing advancement of deep learning recommendation systems.

This paper [130] explores the integration of software and hardware design principles to optimize the performance of LLMs. It focuses on the co-design approach, which synchronizes the development of software algorithms and hardware architecture to achieve efficient processing and better utilization

 doi:10.20944/preprints202411.0156.v1

of resources. This synergy is essential in managing the computational demands of LLMs, which require significant processing power and memory bandwidth. The authors discuss various strategies for optimizing both hardware (such as specialized accelerators and memory hierarchies) and software (such as algorithmic improvements and efficient coding practices) to enhance the performance and efficiency of LLMs. Furthermore, the paper delves into the application of this co-design methodology in design verification processes. Design verification, a critical phase in the development of digital systems, benefits from the enhanced capabilities of co-designed LLMs. By leveraging optimized LLMs, verification tools can process complex datasets and simulations more effectively, leading to more accurate and faster verification results. The integration of co-designed LLMs into verification workflows helps in identifying design flaws early, reducing the time and cost associated with the design and development of hardware systems. The paper highlights case studies and experimental results that demonstrate the practical benefits and improvements achieved through the software/hardware co-design approach in real-world verification scenarios.

This paper [131] investigates the potential of leveraging LLMs in the co-design process of software and hardware, specifically for designing Compute-in-Memory (CiM) Deep Neural Network (DNN) accelerators. It explores how LLMs can be utilized to enhance the co-design process by providing advanced capabilities in generating design solutions and optimizing both software and hardware components concurrently. The authors emphasize the importance of LLMs in automating and improving the design workflow, leading to more efficient and effective development of CiM DNN accelerators. The paper presents a detailed case study demonstrating the application of LLMs in the co-design of CiM DNN accelerators. Through this case study, the authors illustrate how LLMs can aid in identifying optimal design configurations and addressing complex design challenges. The study shows that LLMs can significantly reduce the time and effort required for design iterations and verification, thereby accelerating the overall development process. The findings suggest that integrating LLMs into the co-design framework can result in substantial performance gains and resource savings, highlighting the viability and benefits of using LLMs in the co-design of advanced hardware accelerators.

2.2.3. Hardware Accelerators

This paper [132] presents a comprehensive dataset specifically designed to facilitate the generation of AI accelerators driven by LLMs. The dataset includes a diverse range of hardware design benchmarks, synthesis results, and performance metrics. The goal is to provide a robust foundation for training LLMs to understand and optimize hardware accelerators effectively. By offering detailed annotations and a variety of design scenarios, the dataset aims to enhance the ability of LLMs to generate efficient and optimized hardware designs. The authors detail the structure of the dataset, which covers various aspects of hardware accelerator design, including computational kernels, memory hierarchies, and interconnect architectures. They also discuss the potential applications of the dataset in training LLMs for tasks such as design space exploration, performance prediction, and design optimization. The paper demonstrates the utility of the dataset through several case studies, showing how LLMs can leverage the provided data to generate and optimize hardware accelerators with significant improvements in performance and efficiency.

This study [110] explores the use of LLMs to detect and optimize code for hardware acceleration. The authors propose a methodology where LLMs analyze software codebases to identify sections that can benefit from hardware acceleration. The LLMs are trained to recognize patterns and structures within the code that are amenable to acceleration, such as loops and parallelizable tasks. The paper presents an evaluation of this methodology using several open-source projects, demonstrating that LLMs can effectively identify and suggest optimizations for hardware acceleration. The authors also highlight the potential for integrating this approach into existing development workflows, allowing for seamless detection and acceleration of critical code sections. The study concludes by discussing

 doi:10.20944/preprints202411.0156.v1

the challenges and future directions for improving the accuracy and applicability of LLM-driven code detection for hardware acceleration.

Table 4. Comparison of papers focuses on Hardware Accelerators with LLMs

Paremeter	Approach	References
Objective and Focus	Developing a comprehensive dataset to support LLM-driven AI accelerator generation.	[132]
	Detecting code patterns suitable for hardware acceleration using LLMs.	[110]
	Automating hardware accelerator design using LLMs.	[133]
	Optimizing batched LLM inferencing with a heterogeneous acceleration approach combining NPUs and PIMs.	[134]
	Optimizing memory management and reduce compilation times for multi-core AI accelerators targeting large language models using a hybrid SPM-cache architecture.	[135]
Approach and Methodology	Curating a diverse set of hardware design examples and specifications for LLMs.	[132]
	Training LLMs on a corpus of annotated code examples to detect hardware-accelerable code.	[110]
	Using LLMs to interpret high-level hardware design specifications and generate accelerators.	[133]
	Integrating NPUs and PIMs to handle computation-intensive and memory-bound tasks.	[134]
	Integrating a shared cache with AI cores and employs TMU for cache management, along with tile-level hardware prefetching and dead block prediction.	[135]
Evaluation and Result	Evaluating dataset by the performance of LLMs in generating accurate hardware designs; improvements noted.	[132]
	Measuring accuracy of LLMs in detecting acceleratable code sections and performance gains; significant improvements found.	[110]
	Comparing LLM-generated accelerators with manually designed ones; LLM designs show comparable or superior performance.	[133]
	Benchmarking against CPU and GPU setups; significant improvements in speed and energy efficiency.	[134]
	The system outperforms traditional SPM in mixed-precision quantization scenarios	[135]
Innovation and Impact	First standardized dataset for LLM and hardware accelerator design intersection; potential to advance the field.	[132]
	Application of LLMs to code optimization for hardware acceleration; automates optimization process.	[110]
	Automates traditionally manual hardware design process, reducing development time and cost.	[133]
	Combines NPU and PIM technologies to optimize LLM inferencing; addresses computational and memory challenges.	[134]
	The hybrid SPM-cache architecture introduces novel hardware-level cache management for AI accelerators, especially beneficial for LLMs.	[135]
Future Directions	Expand and diversify the dataset, enhance LLM capabilities for complex tasks.	[132]
	Develop more sophisticated models, integrate with different hardware platforms, expand dataset.	[110]
	Refine models, expand applicability to different accelerators, integrate with design tools.	[133]
	Refine NPU and PIM integration, explore other heterogeneous configurations, expand to other AI workloads.	[134]
	Further optimization of cache replacement policies and better integration of this architecture into future AI accelerator designs for large-scale AI models.	[135]

Gen-acceleration [133] focuses on the pioneering efforts to use LLMs for the automatic generation of hardware accelerators. The paper outlines a novel framework where LLMs are employed to translate high-level design specifications directly into hardware accelerator designs. The approach leverages the NLP capabilities of LLMs to understand and interpret complex design requirements and convert them into efficient hardware architectures. The authors provide a detailed analysis of the framework, including its architecture, training process, and performance evaluation. They demonstrate the effectiveness of the approach through multiple case studies, showing that LLMs can generate hardware accelerators that meet or exceed the performance of manually designed counterparts. The paper also discusses the potential for this technology to revolutionize the hardware design process, making it more accessible and efficient.

NeuPIMs [134] introduces a heterogeneous acceleration framework that combines Neural Processing Units (NPUs) with Processing-In-Memory (PIM) technologies to enhance the performance of batched LLM inferencing. The paper discusses the architectural innovations that enable this combination, focusing on how NPUs and PIM can work together to overcome the memory bandwidth limitations and computational bottlenecks typically associated with LLM inferencing. The authors provide a comprehensive evaluation of the NeuPIMs framework, highlighting its performance benefits across various LLM benchmarks. They demonstrate significant improvements in throughput and energy efficiency compared to traditional GPU-based solutions. The paper also delves into the technical details of the NPU-PIM integration, including the data flow, memory management, and synchronization mechanisms that enable efficient batched inferencing.

The paper [135] presents the LCM (LLM-focused Hybrid Scratch-Pad Memory (SPM)-cache) architecture designed for multi-core AI accelerators to address the growing computational demands of LLMs. By integrating a hybrid system combining SPM and a shared cache, this architecture

 doi:10.20944/preprints202411.0156.v1

provides shorter compilation times and better memory management, particularly for LLMs using mixed-precision quantization. The proposed system utilizes a Tensor Management Unit (TMU) for efficient cache handling and employs innovative hardware prefetching and dead block prediction strategies to mitigate memory access issues. The system outperforms conventional SPM-based architectures, showing up to 50.5% performance improvements in specific scenarios. Table 4 provides an overview of all the papers discussed in the hardware accelerators subsection for comparison.

2.2.4. Hardware Security

DIVAS (Distributed Intelligence for Verification and Security) [136] introduces a comprehensive framework utilizing LLMs to improve the security of System on Chip (SoC) designs. DIVAS integrates multiple security analysis tools and LLMs to provide an end-to-end solution for detecting vulnerabilities and enforcing security policies in SoC designs. The framework automates the identification of security threats and applies policy-based protection mechanisms, with the aim of streamlined and fortifying the security analysis process. DIVAS employs LLMs for various tasks such as vulnerability assessment, anomaly detection, and generating mitigation strategies. The authors detail the system architecture, including its integration with existing SoC design tools and the use of LLMs to interpret and analyze complex security data. The experimental results demonstrate the effectiveness of DIVAS in identifying security vulnerabilities and implementing policies, leading to a significant improvement in SoC security. The paper concludes with a discussion of potential improvements, such as the incorporation of real-time monitoring and refinement LLMs to handle a wider range of security scenarios.

This paper [137] addresses the potential pitfalls of relying on LLMs for hardware specification, particularly in the context of security. The authors argue that, while LLMs can accelerate the specification process, they often generate specifications that are syntactically correct but semantically flawed, leading to security vulnerabilities. The paper presents case studies where LLM-generated specifications resulted in security issues, emphasizing the need for formal methods to verify these specifications. The authors propose a hybrid approach that combines LLMs with formal verification techniques to ensure the correctness and security of hardware specifications. They present a framework that uses LLMs to generate initial specifications, followed by formal verification tools to validate and correct these specifications. Experimental results show that this approach significantly reduces the incidence of security flaws compared to using LLMs alone. The paper concludes with a discussion on the limitations of current LLMs in understanding complex security requirements and the importance of integrating formal methods to achieve reliable and secure hardware designs.

This paper [138] also explores the use of LLMs to identify and fix security bugs in hardware designs. The authors describe a system where LLMs analyzes hardware code to detect potential security vulnerabilities and suggest fixes. This approach aims to automate the bug-fixing process, reducing the time and expertise required to secure hardware designs. The paper details the methodology for training LLMs on hardware security datasets, the types of bugs the system can identify, and the accuracy of the suggested fixes. Experimental results indicate that LLMs can effectively identify and propose solutions for a wide range of security bugs, though their effectiveness varies with the complexity of the bugs. The authors discuss the limitations of LLMs in understanding intricate hardware interactions and suggest future work to improve the robustness of the models and expand their applicability to more complex security scenarios.

This paper [139] examines the potential misuse of general-purpose LLMs in designing hardware Trojans, malicious circuits embedded in hardware designs. The authors demonstrate how LLMs, typically used for benign purposes, can be repurposed to create sophisticated Trojans that are difficult to detect. The paper presents a series of experiments where LLMs are used to generate Trojan designs and assesses their effectiveness and stealthiness. The authors highlight the risks posed by the accessibility of powerful LLMs and the need for robust detection mechanisms. They propose countermeasures, including enhanced verification processes and the development of specialized

 doi:10.20944/preprints202411.0156.v1

LLMs trained to recognize and flag suspicious patterns in hardware designs. The paper concludes by discussing the ethical implications of LLMs in hardware design and the importance of proactive measures to prevent their misuse in creating security threats.

LLM for SoC Security [140] explores the transformative impact of LLMs on the security of SoC designs. The paper argues that LLMs represent a significant advancement in the ability to analyze and secure SoC architectures. The authors describe various applications of LLMs in SoC security, including vulnerability detection, threat modeling, and automated patch generation. The paper provides detailed case studies demonstrating the effectiveness of LLMs in enhancing SoC security. Experimental results show that LLMs can identify vulnerabilities more efficiently and accurately than traditional methods. The authors discuss the challenges in integrating LLMs with existing security workflows and propose solutions to address these challenges. The paper concludes by highlighting the potential for LLMs to redefine SoC security practices and the need for continued research to fully realize their potential.

This paper [141] discusses the dual role of LLMs in chip design: as tools for improving design efficiency and as potential sources of security risks. The authors examine the capabilities of LLMs in automating various aspects of chip design, including specification, verification, and optimization. They also highlight the security risks associated with LLM-generated designs, such as the inadvertent introduction of vulnerabilities and the potential for malicious use. The authors propose a framework for building trust in LLM-generated designs by incorporating rigorous security checks and validation processes. They present case studies where LLMs successfully aided in chip design and instances where they introduced security risks. The paper concludes with recommendations for developing secure LLM workflows, including enhanced training protocols and collaboration between LLM developers and security experts to mitigate risks and ensure the reliability of LLM-assisted chip designs.

This paper [142] investigates the use of LLMs to assist in fixing security bugs in hardware code. The authors present a system where developers can interactively prompt LLMs to identify and correct security vulnerabilities in hardware designs. The approach aims to leverage the language understanding capabilities of LLMs to enhance the efficiency and accuracy of bug fixing. The paper details the interactive prompting system, the types of security bugs it can address, and the effectiveness of the LLM-generated fixes. Experimental results show that the system can significantly reduce the time required to identify and fix security bugs, though the success rate varies with the complexity of the bugs. The authors discuss the limitations of current LLMs in handling complex hardware security scenarios and suggest future research directions to improve the system's robustness and expand its capabilities.

This paper [143] explores the use of LLMs to generate security assertions for hardware designs. Security assertions are critical for verifying that hardware operates as intended without vulnerabilities. The authors propose a framework where LLMs are trained to understand the hardware specifications and automatically generate the corresponding security assertions. The paper demonstrates the effectiveness of this approach through multiple case studies, showing that LLM-generated assertions can identify security issues early in the design process. The authors also discuss the challenges of training LLMs to understand complex hardware specifications and the need for continuous improvement in the training datasets. The study concludes that LLMs has a significant potential to enhance the security verification process by providing accurate and automated security assertions.

 doi:10.20944/preprints202411.0156.v1

Table 5. Comparison of papers focuses on Hardware Security with LLMs

Paremeter	Approach	References
Objective and Focus	Providing comprehensive LLM-based frameworks that enhance security analysis in SoC design by automating tasks such as bug fixing, vulnerability detection, and policy generation.	[136,140,144,145]
	Specific challenges of detecting and fixing bugs in hardware code, as well as the potential misuse of LLMs for malicious purposes like designing hardware Trojans.	[138,139,142]
	Emphasizing the risks of relying on LLM-generated specifications and assertions and advocate for integrating LLMs with formal verification methods to ensure correctness and security.	[137,143]
Approach and Methodology	Utilizing LLMs in a broad range of security tasks, from HDL generation and verification to vulnerability detection and policy enforcement. SoCureLLM stands out for its scalability and focus on large-scale SoC designs.	[136,140,144,145]
	[137] advocates for combining LLMs with formal verification techniques, while [143] focuses on using LLMs to generate security assertions.	[137,143]
	Exploring how LLMs can assist in identifying and fixing hardware security bugs, presenting frameworks that analyze hardware code for vulnerabilities.	[138,142]
Evaluation and Result	Demonstrating significant improvements in detecting vulnerabilities and generating security policies through case studies and experiments, particularly with SoCureLLM outperforming traditional methods in large-scale SoC designs.	[136,140,144,145]
	Showing that combining LLMs with formal methods and generating security assertions can enhance the security of hardware specifications.	[137,143]
	Presenting empirical evidence of LLMs effectively fixing hardware security bugs, though results indicate varying effectiveness depending on the complexity of the bugs.	[138,142]
Innovation and Impact	[144] is unique in addressing scalability issues and applying LLMs to large-scale designs, setting a new standard for hardware security verification frameworks.	[144]
	Pioneering in integrating LLMs for comprehensive security analysis and policy enforcement in SoC designs.	[136,140]
	[138,142] showcase innovative methods for automating hardware bug fixing, while [137] proposes integrating formal methods to avoid the semantic errors associated with LLM-generated specifications.	[137,138,142]
	Raising critical concerns about the potential misuse of LLMs and suggests countermeasures.	[139]
Future Directions	Emphasizing refining LLM integration and expanding their applicability to larger designs and real-time scenarios.	[136,144,145]
	Recommending improving the robustness of LLMs and expanding their applicability to more complex scenarios.	[138,142]
	Continuing to advocate for the integration of formal verification techniques to mitigate the risks posed by LLM-generated code.	[137]

This paper [145] explores how LLMs can be leveraged to enhance SoC security by automating the HDL generation process, bug fixing, and security verification. It highlights the potential of LLMs such as GPT, ChatGPT, and BERT to assist in SoC security through NLP and contextual reasoning. The authors emphasize the role of LLMs in automating the detection of security vulnerabilities across the complex design process of SoCs, offering automated fixes and providing a framework for generating secure HDL code. The paper focuses on the importance of policies and asserts that LLMs can not only identify vulnerabilities but also automatically generate assertions and security policies to mitigate these risks. The paper illustrates several use cases, such as detecting vulnerabilities from HDL documentation and using LLMs to map security weaknesses to known vulnerabilities like Common Weakness Enumerations (CWEs). It also outlines the challenges of applying LLMs to hardware design, such as token limitations and incomplete context processing. The paper concludes by identifying current limitations and suggests directions for improving LLM integration in SoC security.

SoCureLLM [144] introduces an innovative LLM-based framework that addresses the limitations of contemporary hardware security verification techniques, especially their adaptability and scalability for large-scale SoC designs. The framework leverages LLMs' advanced reasoning and pattern recognition abilities to detect vulnerabilities in large SoC designs and generate comprehensive security policies. Unlike previous methods that focus on smaller designs, SoCureLLM overcomes challenges related to token limitations and memory constraints, effectively partitioning designs into smaller code snippets and analyzing them contextually. In evaluations, SoCureLLM successfully detected 76.47% of security bugs on vulnerable RISC-V SoCs and formulated 84 novel security policies. The framework introduces automation in both the detection and the generation of security policies based on threat models, significantly reducing manual effort. The authors emphasize the importance of automated security policy databases to address SoC vulnerabilities and enhance presilicon verification. The paper concludes by highlighting the scalability and effectiveness of SoCureLLM in verifying large designs while proposing potential refinements for future iterations. Table 5 compares all the paper discussed in this subsection.

 doi:10.20944/preprints202411.0156.v1

I have the following Shift Register Written in Verilog:

```
module shift_register #(parameter WIDTH = 8) (
   // Ports omitted for space
);
   reg [WIDTH-1:0] internal_data;

   // Shift register operation
   always @(posedge clk) begin
      if (rst) begin
         internal_data <= {WIDTH{1'b0}};
      end else if (enable) begin
         internal_data <= data_in;
      end else if (scan_enable) begin
         internal_data <= {internal_data[WIDTH-2:0], scan_in};
      end
   end

   // Output assignment
   assign data_out = internal_data;
   assign scan_out = internal_data[WIDTH-1];
endmodule
```

Unfortunately, I doesn't work if the WIDTH is set to 1. Can you fix it?

Figure 8. Input prompt for a HDL debugging

 doi:10.20944/preprints202411.0156.v1

The issue with your shift register occurs when `WIDTH` is set to 1, making the range `internal_data[WIDTH-2:0]` invalid because it results in a negative index. To fix this, you need to handle the case where `WIDTH` is 1 separately.

Here's the modified version of the code that works when `WIDTH` is 1:

verilog Copy code

```
module shift_register #(parameter WIDTH = 8) (
    // Ports omitted for space
);
    reg [WIDTH-1:0] internal_data;

    // Shift register operation
    always @(posedge clk) begin
        if (rst) begin
            internal_data <= {WIDTH{1'b0}};
        end else if (enable) begin
            internal_data <= data_in;
        end else if (scan_enable) begin
            if (WIDTH == 1) begin
                internal_data <= scan_in;
            end else begin
                internal_data <= {internal_data[WIDTH-2:0], scan_in};
            end
        end
    end

    // Output assignment
    assign data_out = internal_data;
    assign scan_out = internal_data[WIDTH-1];
endmodule
```

Key Changes:

- I added a conditional block to handle the case where `WIDTH == 1`. Instead of attempting to access a non-existent bit slice `internal_data[WIDTH-2:0]`, it simply assigns `scan_in` directly to `internal_data`.

With this change, the shift register will now work properly even when `WIDTH` is set to 1.

Figure 9. ChatGPT output for the input prompt of the HDL debugging

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 November 2024 doi:10.20944/preprints202411.0156.v1

The issue arises when the WIDTH is set to 1 because the bit slicing `internal_data[WIDTH-2:0]` becomes invalid. This is because the index `WIDTH-2` would be -1, which is not a valid index for a 1-bit register.

To address this, we can use a conditional statement to handle the case where WIDTH is 1 separately. Here's the modified code:

Verilog

```
module shift_register #(parameter WIDTH = 8) (
    // Ports omitted for space
);
    reg [WIDTH-1:0] internal_data;

    // Shift register operation
    always @(posedge clk) begin
        if (rst) begin
            internal_data <= {WIDTH{1'b0}};
        end else if (enable) begin
            internal_data <= data_in;
        end else if (scan_enable) begin
            if (WIDTH == 1) begin
                internal_data <= scan_in;
            end else begin
                internal_data <= {internal_data[WIDTH-2:0], scan_in};
            end
        end
    end

    // Output assignment
    assign data_out = internal_data;
    assign scan_out = internal_data[WIDTH-1];
endmodule
```

Use code with caution.

With this modification, the code will work correctly even when WIDTH is set to 1. The conditional statement checks if WIDTH is 1 and assigns `scan_in` directly to `internal_data` in that case. Otherwise, it uses the original bit slicing approach.

Figure 10. Gemini output for the input prompt of the HDL debugging

 doi:10.20944/preprints202411.0156.v1

Copilot

The issue arises because when `WIDTH` is set to 1, the expression `internal_data[WIDTH-2:0]` becomes invalid. To fix this, you can add a conditional check to handle the case when `WIDTH` is 1. Here's the modified code:

```
module shift_register #(parameter WIDTH = 8) (
    input clk,
    input rst,
    input enable,
    input scan_enable,
    input [WIDTH-1:0] data_in,
    input scan_in,
    output [WIDTH-1:0] data_out,
    output scan_out
);
    reg [WIDTH-1:0] internal_data;

    // Shift register operation
    always @(posedge clk) begin
        if (rst) begin
            internal_data <= {WIDTH{1'b0}};
        end else if (enable) begin
            internal_data <= data_in;
        end else if (scan_enable) begin
            if (WIDTH == 1) begin
                internal_data <= scan_in;
            end else begin
                internal_data <= {internal_data[WIDTH-2:0], scan_in};
            end
        end
    end

    // Output assignment
    assign data_out = internal_data;
    assign scan_out = internal_data[WIDTH-1];
endmodule
```

This modification ensures that when `WIDTH` is 1, the shift operation is handled correctly. Let me know if this works for you!

Figure 11. Copilot output for the input prompt of the HDL debugging

2.2.5. Hardware Debugging

In Figure 8, we demonstrate an HDL debugging for a shift register, where the input prompt describes the desired behavior and adjustments. The results of this input prompt processed by different LLMs are presented in Figure 9, Figure 10, and Figure 11, showcasing the outputs from ChatGPT 4o, Gemini, and Copilot, respectively. These figures illustrate the varying approaches taken by each model to interpret and modify the HDL design.

HDLdebugger [146] presents a novel approach to debugging HDL code using LLMs. The authors introduce a system that leverages the capabilities of LLMs to automate and streamline the debugging process for HDL, which includes languages like Verilog and VHDL. The system is designed to identify syntax and semantic errors, suggest corrections, and provide explanations for detected issues, thereby reducing the manual effort and expertise required in traditional debugging methods. The paper details the architecture of HDL debugger, which includes modules for parsing HDL code, generating debugging suggestions, and integrating user feedback to refine its outputs. The authors provide experimental results showing that HDL debugger can effectively identify and correct a wide range of common errors in HDL code, significantly improving debugging efficiency. The study concludes by discussing potential enhancements, such as expanding the system's knowledge base and incorporating more advanced ML techniques to handle more complex debugging scenarios.

 doi:10.20944/preprints202411.0156.v1

LLM4SecHW [147] explores the use of a domain-specific LLM for hardware debugging, particularly focusing on security-related issues. The paper introduces a specialized LLM trained on hardware design and security datasets to assist in identifying and fixing security vulnerabilities in hardware designs. The system aims to enhance the debugging process by providing targeted suggestions and fixes for security bugs, which are often more challenging to detect and resolve than general errors. The authors describe the development of LLM4SecHW, including the data collection, training process, and the integration of the model into existing hardware debugging workflows. Experimental results demonstrate that LLM4SecHW is effective in detecting and resolving a variety of security-related issues in hardware designs, outperforming general-purpose LLMs in both accuracy and relevance. The paper concludes with a discussion on the limitations of the current model and future directions for improving its performance, such as expanding the training dataset and refining the model's understanding of complex security scenarios. Table 6 compares the papers discussed in this subsection.

Table 6. Comparison of papers focuses on Hardware Debugging with LLMs

Paremeter	Approach	References
Objective and Focus	Focusing on general HDL debugging, aiming to automate the identification and correction of syntax and semantic errors in HDL code.	[146]
	Targeting hardware debugging with a specific emphasis on security-related issues, leveraging a domain-specific LLM trained on hardware security data.	[147]
Approach and Methodology	Using a general-purpose LLM adapted for HDL debugging, with modules for parsing code, generating suggestions, and integrating user feedback.	[146]
	Employing a specialized LLM trained specifically on hardware security datasets, providing targeted debugging assistance for security vulnerabilities.	[147]
Evaluation and Result	Showing effectiveness in identifying and correcting a wide range of common HDL errors, demonstrating significant improvements in debugging efficiency.	[146]
	Demonstrating superior performance in detecting and resolving security-related issues in hardware designs compared to general-purpose LLMs, highlighting its accuracy and relevance in security contexts.	[147]
Innovation and Impact	Integrating LLMs into the general HDL debugging process, reducing manual effort and expertise required for traditional debugging.	[146]
	Focusing on security-specific hardware debugging, addressing the more complex and critical aspect of hardware design vulnerabilities.	[147]
Future Directions	Expanding the system's knowledge base and incorporating advanced machine learning techniques to handle more complex debugging scenarios.	[146]
	Enhancing the model's performance by expanding the training dataset and refining its understanding of complex security scenarios.	[147]

 doi:10.20944/preprints202411.0156.v1

Provide a UVM verification code for this Verilog design:

```
module shift_register #(parameter WIDTH = 8) (
    // Ports omitted for space
);
    reg [WIDTH-1:0] internal_data;

    // Shift register operation
    always @(posedge clk) begin
        if (rst) begin
            internal_data <= {WIDTH{1'b0}};
        end else if (enable) begin
            internal_data <= data_in;
        end else if (scan_enable) begin
            if (WIDTH == 1) begin
                internal_data <= scan_in;
            end else begin
                internal_data <= {internal_data[WIDTH-2:0], scan_in};
            end
        end
    end

    // Output assignment
    assign data_out = internal_data;
    assign scan_out = internal_data[WIDTH-1];
endmodule
```

Figure 12. Input prompt for a HDL verification

 doi:10.20944/preprints202411.0156.v1

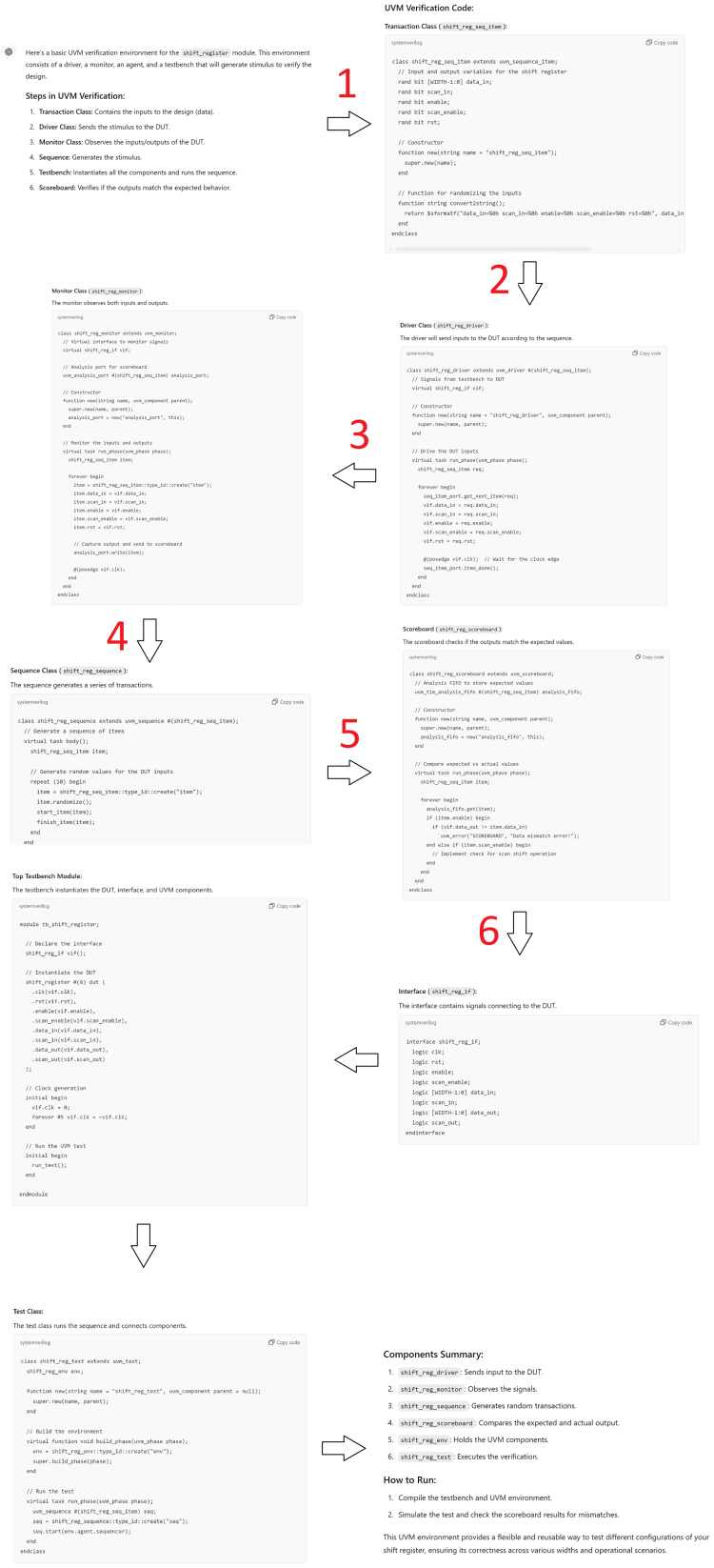

Figure 13. ChatGPT output for the input prompt of the HDL verification

 doi:10.20944/preprints202411.0156.v1

I'd be glad to provide a UVM verification code for the Verilog shift register design you described:

Verilog

```
`timescale 1ns/1ns

module shift_register_tb;

// Parameters
parameter WIDTH = 8;

// Signals
reg clk;
reg rst;
reg enable;
reg scan_enable;
reg scan_in;
reg [WIDTH-1:0] data_in;
wire [WIDTH-1:0] data_out;
wire scan_out;

// DUT instantiation
shift_register #(WIDTH) dut (
    .clk(clk),
    .rst(rst),
    .enable(enable),
    .scan_enable(scan_enable),
    .scan_in(scan_in),
    .data_in(data_in),
    .data_out(data_out),
    .scan_out(scan_out)
);

// UVM environment
`include "uvm_macros.svh"

class shift_register_env extends uvm_env;
  `uvm_component_param(clk_generator, clk)
  `uvm_component_param(rst_generator, rst)
  `uvm_component_param(data_generator, data_in)
  `uvm_component_param(scan_generator, scan_in)
  `uvm_component_param(checker, data_out, scan_out)

  function new(string name = "shift_register_env");
    super.new(name);
  endfunction

  virtual task run_phase(uvm_phase phase);
    if (phase == uvm_phase::run) begin
      fork
        clk_generator.run();
        rst_generator.run();
        data_generator.run();
        scan_generator.run();
        checker.run();
      join
    end
```

1

```
  endtask
endclass

// UVM components
class clk_generator extends uvm_sequence_item;
  `uvm_field_int(clk_period, 10)

  function new(string name = "clk_generator");
    super.new(name);
  endfunction

  task generate(uvm_sequence_item item);
    forever begin
      clk <= 1'b0;
      #clk_period/2;
      clk <= 1'b1;
      #clk_period/2;
    end
  endtask
endclass

// ... (other UVM components for rst_generator, data_generator, scan_ge

// Testbench main
initial begin
  uvm_config_db#(uvm_sequence_item)::set(null, "*", "seq_file", "shift_
  uvm_top::run();
end

endmodule
```

2

Use code with caution.

This UVM verification code provides a comprehensive framework for testing the shift register design, including:

- **UVM environment:** Manages the overall simulation environment and coordinates the execution of various components.
- **UVM components:**
  - `clk_generator`: Generates a clock signal with a specified period.
  - `rst_generator`: Generates a reset signal.
  - `data_generator`: Generates data input sequences.
  - `scan_generator`: Generates scan input sequences.
  - `checker`: Verifies the correctness of the data output and scan output.
- **Testbench main:** Initializes the UVM environment and starts the simulation.
- **Sequence files:** Define the specific test sequences to be executed (not included in the provided code).

You can customize the test sequences in the `shift_register_seq.sv` file to cover different scenarios and test cases for your shift register design.

Figure 14. Gemini output for the input prompt of the HDL verification

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 4 November 2024 doi:10.20944/preprints202411.0156.v1

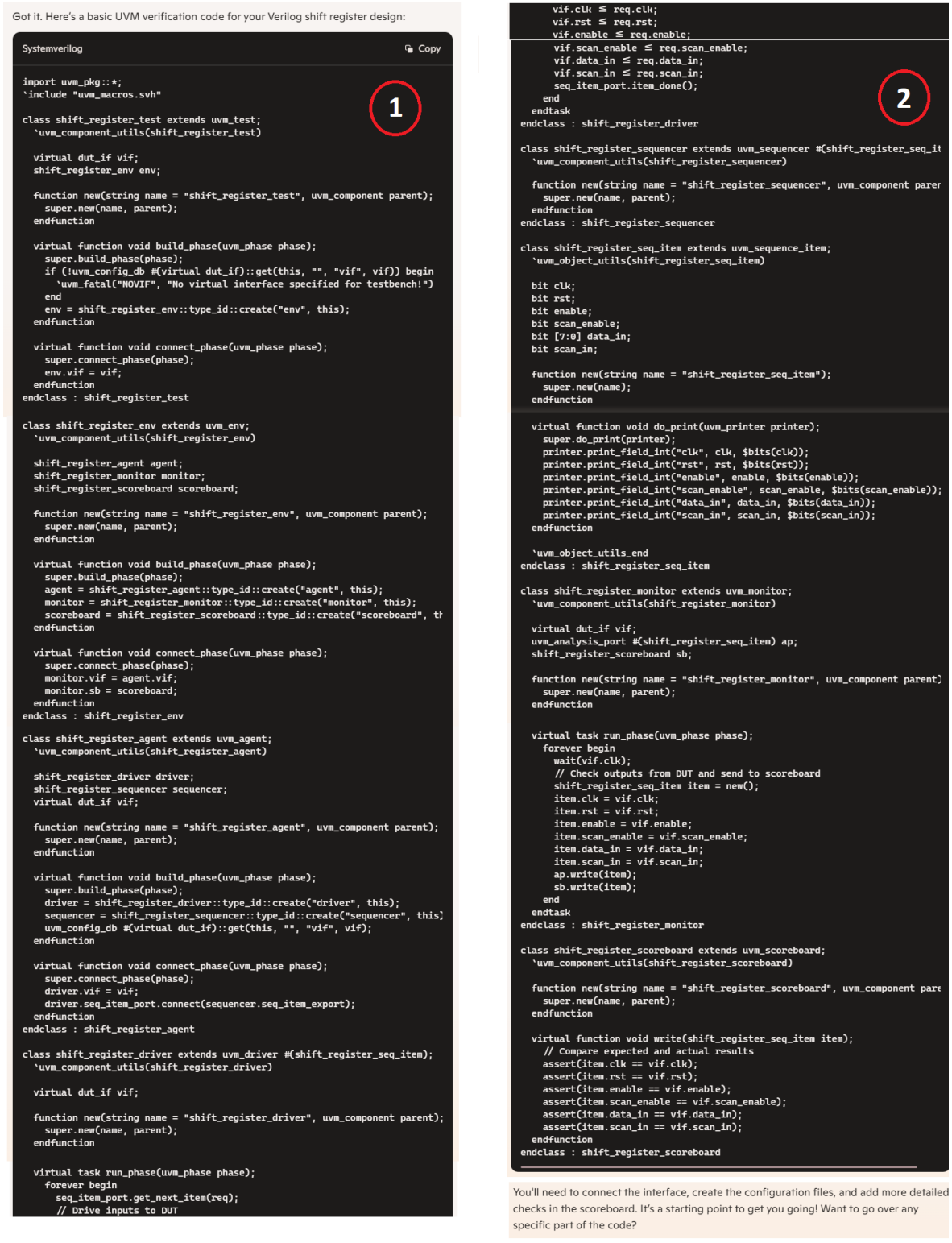

Figure 15. Copilot output for the input prompt of the HDL verification

2.2.6. Hardware Verification

In Figure 12, we demonstrate an HDL verification for a shift register, where the input prompt describes the desired behavior and adjustments. The results of this input prompt processed by different LLMs are presented in Figure 13, Figure 14, and Figure 15, showcasing the outputs from ChatGPT 4o,

 doi:10.20944/preprints202411.0156.v1

Gemini, and Copilot, respectively. These figures illustrate the varying approaches taken by each model to interpret and modify the HDL design.

AssertLLM [148] focuses on utilizing multiple LLMs to generate and evaluate hardware verification assertions directly from the design specifications. It presents a novel approach where various LLMs are used to understand the design specifications and subsequently generate appropriate verification assertions that ensure the correctness of hardware designs. This methodology aims to streamline the process of hardware verification by automating the generation of assertions, which traditionally requires significant manual effort and expertise. The authors detail the framework architecture, which includes parsing design specifications, employing multiple LLMs for assertion generation, and using evaluation metrics to assess the quality and accuracy of the generated assertions. The paper showcases experimental results indicating that the multi-LLM approach produces assertions that are both accurate and relevant, reducing the overall time and effort required for hardware verification. The study concludes by discussing potential improvements and future directions, such as refining LLM training datasets and integrating the framework with existing verification tools.

This paper [149] explores the application of LLMs to assist in the formal verification of RTL designs. It introduces methods for leveraging LLMs to automatically generate properties and invariants that are crucial for formal verification processes. The goal is to improve the efficiency and accuracy of formal verification by reducing the dependency on manual property generation, which can be time consuming and error-prone. The authors present a detailed workflow where LLMs are used to interpret the RTL code and produce formal properties that can be verified using formal methods. They provide experimental results that demonstrate LLM-generated properties can significantly augment the formal verification process, catching more design errors, and reducing verification time. The paper also discusses the limitations of current LLMs in understanding complex RTL semantics and suggests future research directions to improve LLM capabilities in formal verification tasks.

This paper [150] addresses the optimization of design verification processes using ML techniques, presenting an open source solution for the hardware design community. The authors outline a framework that integrates ML algorithms to predict verification outcomes, identify potential design issues early, and prioritize verification tasks. This approach aims to enhance the efficiency and effectiveness of the design verification process by leveraging data-driven insights. The framework's implementation is discussed in detail, including the selection of ML models, training datasets, and integration with existing verification workflows. Experimental results are provided to demonstrate the improvements in verification coverage and reduction in verification time achieved by the ML-enhanced framework. The paper concludes with a discussion on the open-source nature of the solution, encouraging community collaboration and further development, as well as outlining potential future enhancements such as incorporating more sophisticated ML models and expanding the dataset.

VerilogEval [151] focuses on assessing the performance of LLMs in generating Verilog code, which is critical for hardware design. The paper provides a comprehensive evaluation framework to benchmark various LLMs' ability to generate accurate and efficient Verilog code from high-level descriptions. The authors detail the criteria used for the evaluation, including code accuracy, readability, and resource efficiency. The study presents extensive experimental results comparing different LLMs on a series of Verilog code generation tasks. These results highlight the strengths and weaknesses of each model, providing insights into how well they can support hardware designers in automating code generation. The paper concludes by discussing the implications of these findings for future research and development, suggesting ways to improve LLM training for better performance in hardware design applications, such as using more domain-specific datasets and refining model architectures.

This paper [152] proposes a method for enhancing verification productivity by translating natural language descriptions into System Verilog assertions with a circuit-aware approach. The authors introduce a system that leverages NLP techniques to interpret design specifications and automatically generate corresponding verification assertions. This approach aims to bridge the gap between high-level design intents and low-level verification tasks, making the verification process

 doi:10.20944/preprints202411.0156.v1

more intuitive and efficient. The system's architecture is described in detail, including the use of NLP models to understand natural language inputs and generate accurate System Verilog assertions that are contextually aware of the circuit design. Experimental results are presented to demonstrate the effectiveness of the proposed method, showing significant improvements in the speed and accuracy of assertion generation compared to manual methods. The paper concludes with discussions on the potential impact of this approach on the verification industry and future research directions to further refine the translation process.

ChipNeMo [153] explores the adaptation of LLMs specifically for the domain of chip design. The paper presents techniques for fine-tuning LLMs on chip design-related datasets to enhance their performance in tasks such as code generation, optimization, and verification. The goal is to create domain-adapted models that can understand and generate chip design content more effectively than general-purpose LLMs. The authors describe the process of collecting and curating domain-specific datasets, the fine-tuning methodology, and the evaluation metrics used to assess the performance of the adapted models. Experimental results show that ChipNeMo outperforms general-purpose LLMs in various chip design tasks, demonstrating the benefits of domain adaptation. The paper concludes with a discussion on the broader implications of domain-adapted LLMs for the chip design industry and potential future research directions, including the integration of these models into existing design workflows and further refinement of adaptation techniques.

LLM4DV [154] investigates the use of LLMs to generate test stimuli for hardware verification. The paper presents methods for training LLMs to understand hardware design specifications and produce relevant test cases that can be used to verify the correctness and performance of hardware designs. This approach aims to automate the test generation process, reducing the time and effort required for comprehensive hardware verification. The authors provide a detailed description of the training process, including the selection of training data, model architecture, and the generation of test stimuli. Experimental results demonstrate that LLM4DV-generated test cases are effective in identifying design issues and achieving high coverage in verification tasks. The paper discusses the potential for further enhancing test generation capabilities by incorporating more sophisticated LLMs and integrating the approach with existing verification frameworks. It concludes by highlighting the benefits of using LLMs in hardware verification, such as increased efficiency and reduced manual effort.

This paper [155] explores the use of LLMs to assist in generating hardware assertions, which are critical for verifying the correct behavior of hardware designs. The authors present a framework that uses LLMs to interpret design documentation and automatically produce assertions that can be used in formal verification processes. The goal is to streamline the assertion generation process and ensure comprehensive coverage of design specifications. The framework architecture is described, including the use of LLMs to parse design documents and generate assertions, and evaluation methods to assess the quality and precision of these assertions. Experimental results indicate that the LLM-assisted approach produces high-quality assertions that can significantly improve verification efficiency. The paper concludes with a discussion on the limitations of current LLMs in understanding complex hardware designs and potential future research directions to improve the capabilities of LLMs in generating assertions, such as incorporating more detailed training data and refining model architectures. Table 7 compares all the papers discussed in this subsection.

 doi:10.20944/preprints202411.0156.v1

Table 7. Comparison of papers focuses on Hardware Verification with LLMs

Paremeter	Approach	References
Objective and Focus	[148,152,155] focus on generating verification assertions, but [148] uses multiple LLMs for better accuracy.	[148,152,155]
	Focusing on enhancing verification through formal methods and machine learning, respectively.	[149,150]
	Focusing on the generation and evaluation of Verilog code and domain-adapted LLMs for chip design.	[151,153]
	Focusing on generating hardware test stimuli, providing a distinct angle on improving the verification process.	[154]
Approach and Methodology	[148,155] use LLMs to interpret design documents and generate assertions, but [148] emphasizes a multi-LLM approach.	[148,155]
	Automating the generation of properties for formal verification.	[149]
	Using ML techniques rather than purely LLMs to optimize the verification process.	[150]
	Fine-tuning and benchmarking LLMs for specific tasks related to Verilog and chip design.	[151,153]
	Utilizing LLMs for generating test stimuli based on hardware design specifications.	[154]
Evaluation and Result	Improving accuracy and efficiency in assertion generation.	[148,155]
	Enhancing error detection and reducing verification time.	[149]
	Improving the verification coverage and time savings using ML.	[150]
	Highlighting the strengths and weaknesses of different LLMs in Verilog code generation.	[151]
	Showing the benefits of domain adaptation in LLMs for chip design tasks.	[153]
	Providing evidence of effective test case generation, improving coverage and identifying design issues.	[154]
Innovation and Impact	Using multiple LLMs for assertion generation is innovative in its multi model approach.	[148]
	Integrating LLMs into the formal verification process, traditionally a manual task.	[149]
	Providing an open-source solution that encourages community development.	[150]
	Offering a comprehensive benchmarking framework for evaluating LLM performance in Verilog code generation.	[151]
	Emphasizing domain adaptation, showing significant performance improvements in chip design tasks.	[153]
	Focusing on automation in different aspects of the verification process, enhancing efficiency and effectiveness.	[154,155]
Future Directions	Refining LLM training datasets, integrating frameworks with existing tools, and enhancing model architectures.	[148–151,153,155]
	Improving the understanding of complex hardware designs and further adaptation techniques.	[148,153,155]
	Highlighting the need for more sophisticated ML and LLM models to handle complex verification tasks.	[149,150]
	Emphasizing continued benchmarking and adaptation to specific hardware design requirements.	[151,153]
	Integrating more advanced LLMs and expanding test generation capabilities within verification frameworks.	[154]

2.3. Use Cases and Success Stories

LLMs have seen promising applications in real-world hardware design projects, particularly in areas such as Design for Manufacturing (DfM) and embedded system development. LLMs have been integrated into DfM processes to help generate, optimize, and modify designs based on manufacturing constraints. For example, in a study, an LLM was used to modify a parametric design of parts like LEGO bricks and CNC machined components, offering improvements in manufacturability and efficiency. LLM could even suggest optimal manufacturing techniques and improve design geometry to streamline production [156].

In the realm of hardware design, LLMs have also been employed to generate HDL code such as Verilog. For instance, LLMs can be used to write, annotate, and debug hardware code, potentially improving design efficiency and reducing errors. While still in experimental stages, these systems show potential in automating parts of the hardware development process [157].

LLMs have shown promise in generating code for embedded systems, particularly in automating the development of C/C++ code for hardware platforms. While their use is still evolving in this field, LLMs offer potential to make hardware programming more approachable and assist in debugging and improving low-level code [158].

LLMs, such as GPT-3.5 and GPT-4, have been applied to generate and refine Verilog code used in hardware design. A process called "VeriRectify" allows the LLM to take error diagnostics from simulators (e.g., iverilog) and iteratively refine the generated Verilog code, significantly improving its correctness and performance. This approach has led to successful code generation for complex hardware designs like multipliers, incorporating both error correction and PPA checks, ensuring optimal designs [151,159].

 doi:10.20944/preprints202411.0156.v1

LLMs are also being explored for validating system configurations. The "Ciri" framework leverages models like GPT-4 to detect misconfigurations in large-scale systems (e.g., HBase, ZooKeeper). Ciri uses prompt engineering and few-shot learning to improve accuracy, achieving promising F1 scores in identifying configuration errors. While it excels at catching certain types of misconfigurations, challenges remain, such as handling dependency violations [160]

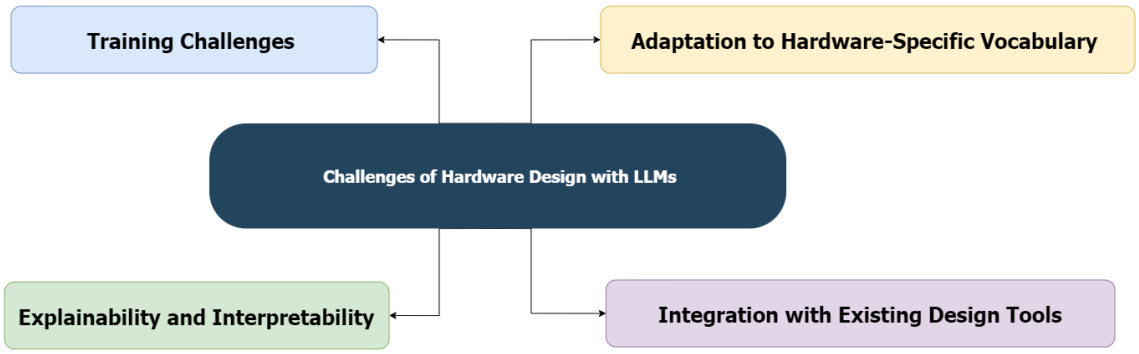

Figure 16. The most notable challenges of Hardware Design with LLMs

3. Challenges

The most important challenges of LLMs in hardware design and verification are shown in Figure 16.

3.1. Training Challenges

Training LLMs for hardware design and verification faces substantial challenges primarily due to the nature of the data and the computational demands involved. Hardware design and verification data are highly specialized, often proprietary, and not readily available in public repositories. This scarcity of high-quality, annotated data makes it difficult to train effective models. Moreover, the data's heterogeneity, encompassing HDL code, schematic diagrams, and natural language documentation, complicates the integration into a cohesive training set. Additionally, the technical expertise required for labeling such data further complicates and prolongs the preparation process. The resource-intensive nature of training LLMs, which demands powerful GPUs or TPUs, significant memory, and extensive storage, is another major hurdle. The energy consumption associated with training these models is also considerable, raising sustainability concerns.

The domain-specific nature of hardware design and verification adds another layer of complexity. LLMs must understand and accurately interpret the intricate language and concepts specific to this field, ensuring that generated outputs are both syntactically correct and semantically meaningful. Errors in generated code can lead to significant functional issues, necessitating robust verification and validation processes. The black-box nature of LLMs presents interpretability challenges, making it difficult to trace and understand their decision-making processes. This lack of transparency can hinder trust and acceptance among hardware engineers, who require clear explanations for the model's outputs. Integrating LLMs into existing workflows without causing disruptions, ensuring compatibility with various industry tools, and providing adequate training and support for users are critical for successful implementation. Addressing these challenges requires a multidisciplinary approach, continuous collaboration, and advances in both ML and hardware design methodologies.

To overcome data scarcity and quality issues, collaboration between academia, industry, and research institutions is crucial. Establishing partnerships can facilitate the sharing of proprietary data under strict confidentiality agreements, expanding the pool of available training data. Additionally, crowdsourcing initiatives and competitions can be organized to gather more annotated data. Developing advanced data augmentation techniques can help generate synthetic data that closely mimics real-world hardware design scenarios, enhancing the diversity and richness of the training

 doi:10.20944/preprints202411.0156.v1

datasets. Implementing semi-supervised or self-supervised learning methods can also leverage unlabeled data to improve model performance.

3.2. Adaptation to Hardware-Specific Vocabulary

Adapting LLMs to hardware-specific vocabulary for hardware design and verification involves enhancing the models' ability to understand and generate text that accurately reflects the specialized terminology and concepts used in this field. This process begins with the careful curation of domain-specific corpora, which include HDLs like VHDL and Verilog, technical documentation, research papers, and industry standards. These corpora provide the foundational data that the models need to learn the syntax, semantics, and context of hardware-related terms. Fine-tuning pre-trained LLMs on these specialized datasets enables the models to grasp the nuances of hardware design language, ensuring that they can generate syntactically correct and semantically meaningful code. Additionally, incorporating glossaries and thesauri specific to hardware design during the training phase can help models better understand and use technical terms accurately.

Furthermore, integrating feedback loops with domain experts during the training and evaluation phases is crucial for refining the models' understanding of hardware-specific vocabulary. Experts can review and provide insights on the models' outputs, identifying areas where the terminology might be used incorrectly or where the context may be misunderstood. This feedback can be used to iteratively improve the models, making them more adept at handling the intricate and precise language of hardware design. Leveraging transfer learning techniques, where models trained on general data are subsequently fine-tuned on hardware-specific data, also enhances their adaptability. This approach allows the models to retain their broad language understanding while becoming proficient in the specialized vocabulary of hardware design and verification. By continuously refining the training process and incorporating expert feedback, LLMs can become powerful tools for assisting in hardware design and verification tasks, offering precise and reliable language understanding tailored to the domain.

3.3. Explainability and Interpretability

Explainability and interpretability are critical issues when applying LLMs to hardware design and verification, given the need for transparency, traceability, and trust in the model's outputs. In hardware design, even minor errors can lead to significant functional flaws, making it essential for engineers to understand how and why an LLM arrived at a particular decision or generated specific code. The black-box nature of LLMs, where the internal workings are opaque and difficult to interpret, poses a significant challenge. This lack of transparency can hinder engineers' ability to trust the model's outputs, as they cannot easily verify the correctness or understand the rationale behind specific decisions. Moreover, in the highly specialized field of hardware design, the contextual nuances and technical details are crucial, and any misinterpretation can have far-reaching consequences.

To address these issues, several strategies can be employed to enhance the explainability and interpretability of LLMs in hardware design and verification. One approach is the use of attention mechanisms and visualization tools that highlight which parts of the input data the model focused on when generating its outputs. This can provide engineers with insights into the model's decision-making process and help them understand how specific terms and concepts were interpreted. Another strategy is incorporating explainable AI (XAI) techniques, which aim to make the outputs of ML models more transparent and interpretable. For instance, generating intermediate representations or step-by-step explanations can help bridge the gap between the model's complex computations and the human engineer's understanding. Additionally, involving domain experts in the training and validation process can ensure that the model's explanations align with the technical realities of hardware design. By continuously refining the model based on expert feedback and focusing on transparency, LLMs can become more reliable and trustworthy tools in the hardware design and verification process.

 doi:10.20944/preprints202411.0156.v1

3.4. Integration with Existing Design Tools

Integrating LLMs with existing design tools in hardware design and verification presents several challenges due to the specialized and complex nature of these tools. Hardware design workflows rely heavily on sophisticated software such as EDA tools, simulation environments, and formal verification systems. These tools have been optimized over years to handle specific tasks efficiently, and introducing LLMs into this ecosystem requires ensuring seamless interoperability. One significant issue is compatibility: the LLMs must be able to interact with various file formats, data structures, and APIs used by these tools. This necessitates the development of robust interfaces and integration layers that can bridge the gap between the general-purpose nature of LLMs and the specialized requirements of hardware design tools.

Additionally, the integration process must ensure that the introduction of LLMs does not disrupt existing workflows. Hardware engineers are accustomed to certain processes and toolchains that have been fine-tuned for productivity and reliability. Introducing LLMs should enhance these workflows by providing added value, such as automating repetitive tasks, offering code suggestions, and assisting in error detection, without requiring significant changes to established practices. This requires careful design of user interfaces and interaction models, ensuring that the outputs from LLMs are presented in a way that is intuitive and easily actionable for engineers. Furthermore, extensive testing and validation are needed to ensure that the LLMs' outputs are reliable and accurate, thereby gaining the trust of the engineers. Training sessions and comprehensive documentation will also be necessary to help users understand and effectively utilize the new capabilities brought by LLMs. By addressing these integration challenges thoughtfully, LLMs can be successfully incorporated into the hardware design and verification ecosystem, enhancing efficiency and innovation.

4. Open Issues

4.1. Unexplored Applications

Table 8. Open issues in terms of main domains, tasks, and LLM usage for future research

Domain	Task	LLM Use
HLS	Automating high-level code to RTL.	Optimizing for performance, area, and power.
HDL Generation	Creating RTL from specifications.	Automating Verilog, VHDL, or SystemVerilog generation.
Component Integration	Managing interactions between hardware modules.	Automating interface generation and integration.
Design Optimization	Improving performance, power, and area iteratively.	Suggesting optimal configurations and design alternatives.
Finite State Machines (FSM) **Design**	Designing FSMs to control hardware modules.	Generating and optimizing FSM transitions and states.
Design Space Exploration	Exploring multiple configurations for performance, power, and area.	Suggesting optimal configurations and trade-offs.
Power-Aware Design	Designing hardware with a focus on power efficiency.	Recommending power-saving techniques like clock gating.
Timing Analysis	Ensuring hardware meets timing constraints.	Optimizing clock trees and fixing timing violations.
Floorplanning	Optimizing the placement of components on a chip.	Assisting in module placement and layout optimization.
Low-Power Design	Implementing low-power design techniques.	Suggesting balanced performance-power trade-offs.
Hardware Accelerators	Designing specialized hardware accelerators.	Creating optimized architectures for AI hardware like GPUs and TPUs.
Clock Tree Synthesis	Creating a balanced clock distribution network.	Optimizing clock tree generation for minimal skew.
Chip Architecture Design	Defining the overall chip architecture and data flow.	Generating architectural suggestions and optimizing data flow.
Physical Layout	Determining how components are placed and routed.	Suggesting efficient routing paths and placements.
ASIC Design	Designing custom integrated circuits.	Automating design optimizations for ASICs.
Fault-Tolerant Design	Creating hardware with built-in redundancy.	Assisting in the creation of error-correcting codes and self-test logic.
Verification Plans	Creating verification plans for hardware.	Generating comprehensive verification plans and test cases.

LLMs have the potential to significantly improve a variety of sub-domains by automating and optimizing complex tasks. LLMs can streamline the HLS process by converting high-level code into RTL code and optimizing for performance, area, and power. Similarly, in HDL Generation, LLMs can automate the generation of Verilog, VHDL, or SystemVerilog based on functional specifications.

Moreover, LLMs can enhance Component Integration by automating the management of hardware module interactions, while in Design Optimization, they can suggest optimal configurations that balance performance and power trade-offs. In areas like FSM design and Design Space Exploration (DSE), LLMs can assist in optimizing state transitions, identifying unreachable states, and exploring multiple design alternatives for better performance.

LLMs are also helpful in power-conscious tasks like Power-Aware Design, providing suggestions for power-saving techniques, and in Timing Analysis and Optimization, where they assist in meeting

 doi:10.20944/preprints202411.0156.v1

timing constraints through clock tree and propagation optimizations. Other physical design tasks, such as floor planning and physical layout, benefit from the ability of LLMs to optimize component placement and routing for improved performance and reduced congestion.

The role of LLMs extends to Low-Power Design Techniques, Hardware Accelerators, and Clock Tree Synthesis (CTS), where they help designers balance power with performance, create architectures for AI accelerators, and optimize clock distribution networks. LLMs can also influence Chip Architecture Design by proposing architectural suggestions and improving data flow.

In advanced design areas such as ASIC Design, Fault-Tolerant Design, and Verification Plan Generation, LLMs provide automated support for optimizing integrated circuits, generating redundancy schemes, and creating comprehensive verification plans. For detailed tasks and applications of LLMs in each sub-domain, refer to Table 8, which offers a structured overview of how LLMs contribute to enhancing various aspects of hardware design.

4.2. Research Gaps

The current research landscape for LLMs in hardware design reveals several significant gaps that limit their application in critical areas such as formal verification, design optimization, and security. Despite the potential of LLMs, their integration with formal methods remains largely absent, posing risks to safety-critical systems. Moreover, the lack of contextual understanding in optimizing complex design trade-offs between PPA hampers multi-objective optimization efforts. Security vulnerabilities specific to hardware designs are also underexplored, leaving hardware systems at risk. The shortage of specialized datasets further compounds the challenge, limiting LLMs' effectiveness in hardware-specific tasks like circuit design and verification. As hardware designs scale in complexity, managing large systems like SoCs becomes increasingly challenging for LLMs, and areas such as analog/mixed-signal design, hardware/software codesign, and post-silicon validation remain underdeveloped. Table 9 provided details on the specific gaps in each of these areas and their respective impacts, offering a clearer picture of the current limitations and opportunities for further research. The most significant sub-domains for each domain can be mentioned as follows:

Table 9. Gaps and Impacts of LLMs in Hardware Design

Category	Gap	Impact
Integration with Formal Methods	LLMs lack integration with formal verification methods.	Risk to safety-critical designs.
Lack of Contextual Understanding for Design Optimizations	LLMs struggle with design trade-offs between PPA.	Multi-objective optimization challenges in hardware design.
Limited Exploration of Hardware Security Vulnerabilities	LLMs are not widely applied to hardware-specific security issues.	Hardware designs remain vulnerable to attacks and misconfigurations.
Inadequate Training Data for Hardware-Specific Tasks	Lack of specialized datasets for hardware design.	LLMs perform poorly on tasks like digital circuit design or corner case verification.
Challenges in Scaling LLMs for Large Hardware Designs	Scaling LLMs for complex hardware like SoCs is difficult.	Full-chip verification is not efficiently managed by current LLM systems.
Underdeveloped Use in Analog and Mixed-Signal Design	Few applications of LLMs in AMS design.	AMS circuits are critical in many systems, and research in this area is lacking.
Lack of Research on Hardware/Software Codesign	Limited research on LLMs for hardware/software optimization.	Co-optimization of hardware and software in SoCs remains unaddressed.
Challenges in Post-Silicon Validation and Debugging	LLMs are not used in post-silicon validation.	Detecting issues after fabrication is not automated by LLM systems.
Limited Explainability and Interpretability in Hardware Design	LLMs often lack clear explanations for their design choices.	Designers lack trust in LLM solutions.
Lack of Efficient DSE	LLMs have not been fully used for DSE.	Optimizing design variants for power, area, and performance remains a challenge.
Minimal Use in Advanced Verification Techniques (UVM, SystemVerilog Assertions)	Research on \gls{uvm} and SystemVerilog Assertions with LLMs is limited.	Verification for complex designs remains unoptimized.
Underdeveloped Role in Fault-Tolerant Hardware Design	Fault-tolerance design using LLMs is unexplored.	Missed opportunity to design reliable systems for industries like aerospace.
Limited Optimization for FPGA Design Automation	LLMs are not widely applied to FPGA design processes like place-and-route.	FPGA design and prototyping are slower without LLM automation.

- **HLS**
  - **Behavioral Synthesis**: Converting behavioral descriptions directly into RTL [161,162].
  - **Scheduling**: Managing operation timing to meet performance and resource constraints [163,164].
  - **Resource Allocation**: Assigning operations to hardware resources such as registers or functional units [165,166].
  - **Loop Unrolling and Pipelining**: Enhance parallelism by optimizing loop handling [167, 168].

 doi:10.20944/preprints202411.0156.v1

  - **Bit-width Optimization**: Minimizing the width of variables without sacrificing accuracy [169].
  - **Control Flow Management**: Managing control flow statements (if-else, switch-case) for hardware synthesis [170].
  - **Memory Access Optimization**: Efficiently handling memory access patterns to reduce latency [171,172].
  - **Interface Generation**: Creating interfaces for communication between blocks during synthesis [173].

- **HDL Generation**
  - **Synthesis-ready HDL Code Generation**: Automatically generating Verilog or VHDL that is ready for synthesis [174].
  - **Parameterized HDL Code**: Creating reusable code with configurable parameters [175].
  - **State Machine Code Generation**: Creating FSMs based on behavioral specifications [176, 177].
  - **HDL Code Translation**: Translating high-level or behavioral code into synthesizable HDL [178,179].
  - **Testbench Generation**: Automatically generating testbenches for HDL verification [180–182].
  - **Hierarchical Module Design**: Automatically generating modular and hierarchical HDL blocks [183].
  - **Assertions and Constraints**: Generating assertion-based HDL for formal verification [148, 184].
  - **Code Formatting and Cleanup**: Ensuring HDL code is clean, well-formatted, and error-free [5] [6].

- **Component Integration**
  - **Interface Synthesis**: Automatically generating interfaces (e.g. Advanced eXtensible Interface (AXI), Advanced Microcontroller Bus Architecture specification (AMBA)) between hardware modules [185].
  - **Signal Mapping**: Automating the signal connection and mapping between modules [186].
  - **Inter-module Communication**: Managing and optimizing data and control flow between different hardware blocks [187].
  - **Bus Arbitration**: Design of efficient bus systems for shared resources [188].
  - **Protocol Handling**: Automating protocol management for communication between modules [7].
  - **System Verification**: Automatically generating testbenches to verify integrated systems [176,189].
  - **Hierarchical Integration**: Building hierarchical hardware designs by integrating reusable IP blocks [190,191].

- **Design Optimization**
  - **Timing Optimization**: Optimizing designs to meet timing constraints [192,193].
  - **Power Optimization**: Minimizing power consumption through design improvements [194–196].
  - **Area Optimization**: Reducing the overall silicon area used by the hardware [197].
  - **Pipelining and Parallelization**: Introducing pipelining and parallelization to improve performance [198,199].
  - **Data Path Optimization**: Optimizing the data path for reduced latency and better resource utilization [200].
  - **Power-Performance Trade-offs**: Balancing between performance gains and power savings [201,202].

---

[5] https://www.einfochips.com/blog/lint-and-cdc-in-vlsi-ensuring-robust-design-and-verification/
[6] https://blogs.sw.siemens.com/verificationhorizons/2021/10/28/hdl-coding-standards-for-do-254/
[7] https://www.mhtechin.com/support/communication-protocol-roadmap-for-embedded-systems-by-mhtechin-a-comprehensive-guide/

 doi:10.20944/preprints202411.0156.v1

  - **Critical Path Reduction**: Shortening the critical path to improve clock frequency [203,204].

- **FSM Design**
  - **State Minimization**: Reducing the number of states to optimize hardware usage [205–207].
  - **Hierarchical FSM Design**: Creating complex FSMs using a hierarchical approach [208].
  - **Error Detection FSMs**: Design of FSMs with built-in error detection mechanisms [209,210].
  - **Power-aware FSM Design**: Creating FSMs optimized for low power consumption [211].
  - **State Encoding Optimization**: Optimizing state encodings (e.g., one-hot, binary) for efficiency [212].
  - **Timing-aware FSM Design**: Ensure that FSMs meet timing constraints [213].
- **DSE**
  - **Pareto-optimal Design Space Exploration**: Exploring the design space to identify Pareto-optimal trade-offs between power, area, and performance [214].
  - **Multi-objective Optimization**: Optimizing designs for multiple conflicting objectives (e.g., power vs. performance) [215].
  - **Parametric Design Exploration**: Exploring various parameter configurations to achieve optimal results [216].
  - **Constraint-driven Design**: Ensure that all design options meet predefined constraints [217].
  - **ML-Assisted DSE**: Using ML models to predict design performance and help in exploration [218–220].
  - **Scenario-based DSE**: Exploring designs based on different use-case scenarios (e.g., high-performance vs. low-power modes) [221].
- **Power-Aware Design**
  - **Dynamic Power Management**: Real-time power adjustments based on workload [201,202,222,223].
  - **Clock Gating**: Reducing power consumption by turning off clocks in idle modules [194,224].
  - **Voltage Scaling**: Dynamically adjust the voltage supply to minimize power usage [225].
  - **Multi-VDD Design**: Design of hardware that operates at multiple voltage levels for different modules [226].
  - **Leakage Power Reduction**: Techniques to reduce leakage power in idle states [227].
  - **Low-power Memory Design**: Optimizing memory access and design to minimize power [228,229].
  - **Thermal-aware Power Optimization**: Ensuring power optimizations also consider thermal constraints [230,231].
- **Timing Analysis and Optimization**
  - **Static Timing Analysis (STA)**: Automatically analyzing and optimizing timing paths [232].
  - **Critical Path Analysis**: Identifying and optimizing the critical path to ensure timing closure [233].
  - **Clock Skew Minimization**: Optimizing the clock distribution to minimize the skew in the design [234].
  - **Multi-clock Domain Design**: Managing and optimizing designs with multiple clock domains [235,236].
  - **Hold and Setup Time Optimization**: Ensure that all paths meet the hold and setup time constraints [237].
  - **Path Delay Optimization**: Shortening the longest paths in the design to improve performance [238].
- **Floorplanning and Physical Design**
  - **Component Placement Optimization**: Place components to minimize delays and area usage [239].
  - **Power Grid Design**: Design of power distribution networks to ensure reliable power delivery [240].
  - **Routing Congestion Management**: Optimize placement to avoid routing congestion and improve performance [241].

 doi:10.20944/preprints202411.0156.v1

  - **Thermal-aware Floorplanning**: Ensure that heat-generating components are placed efficiently for heat dissipation [242,243].
  - **Timing-aware Floorplanning**: Ensure that critical timing paths are optimized in the placement process [244].

- **Low-Power Design Techniques**
  - **Multi-threshold Design**: Using multiple threshold voltages to optimize power [245,246].
  - **Adaptive Voltage Scaling (AVS)**: Dynamically adjust voltage levels for power savings [247,248].
  - **Dynamic Power Gating**: Turn off parts of the circuit during inactivity to save power [249,250].
  - **Energy-efficient Logic Synthesis**: Design circuits with low power consumption from the ground up [251].
  - **Sleep Mode Design**: Implement deep sleep modes for minimal power consumption during idle periods [252].

- **Hardware Accelerators**
  - **AI/ML Accelerators**: Design of hardware specialized for AI/ML tasks (for example, matrix multiplication, deep learning) [253,254].
  - **DSP Accelerators**: Optimizing hardware for DSP tasks such as filtering and transforms [255,256].
  - **Parallel Computing Accelerators**: Designing hardware for parallel task execution, such as multi-core GPUs or FPGAs [257,258].
  - **Memory Bandwidth Optimization**: Ensure that memory subsystems can handle the data bandwidth required for high-throughput hardware accelerators [259,260].
  - **Reconfigurable Accelerators**: Design flexible hardware accelerators, such as FPGAs, that can be adapted to different tasks [261,262].
  - **Low-Power Accelerators**: Optimizing accelerators to be power efficient, especially for mobile and embedded systems [263,264].
  - **AI Hardware/Software Co-optimization**: Co-design of AI software and hardware to maximize hardware utilization and performance [265,266].

- **CTS**
  - **Clock Skew Minimization**: Ensure that clock signals arrive at all components simultaneously to minimize skew [267].
  - **Power-aware CTS**: Design of clock trees to minimize power consumption [268].
  - **Multi-domain Clock Tree Design**: Managing multiple clock domains to ensure efficient clock distribution [269].
  - **Clock Buffer Insertion**: Strategic placement of buffers to reduce clock delay and skew [270].
  - **Gated Clock Design**: Reduction of power by turning off the clock in inactive areas of the design [271,272].
  - **CTS for Low-power Designs**: Techniques to reduce clock power consumption, like multi-threshold designs or clock gating [273].

- **Chip Architecture Design**
  - **Multi-core Architecture Design**: Defining architectures with multiple cores for parallelism and performance improvements [274,275].
  - **Memory Hierarchy Design**: Design of the memory subsystem, including caches, RAM, and register files, to optimize data access speeds [276,277].
  - **Dataflow Architecture**: Optimizing the flow of data between processing units to maximize throughput [278,279].
  - **Instruction Set Architecture (ISA)**: Defining and optimizing the instruction set used by processors [280,281].
  - **On-chip Communication Networks**: Design of networks for communication between cores and other processing units [282–287].

 doi:10.20944/preprints202411.0156.v1

  - **Performance Bottleneck Identification**: Analyze architectural designs to find and eliminate performance bottlenecks [288,289].
  - **Scalability Optimization**: Ensure that the chip architecture scales well with increasing system complexity (e.g., more cores or memory) [290,291].

- **Physical Layout and Routing**
  - **Component Placement Optimization**: Identifying the optimal placement of components to reduce routing length and delay [239,292].
  - **Wire Routing**: Definition of the path of signal wires to minimize delay, power consumption, and congestion [293,294].
  - **Congestion Management**: Preventing over-congested regions in the design by optimizing component placement and routing [295,296].
  - **Thermal-aware Layout**: Ensure that heat-generating components are placed strategically for heat dissipation [297,298].
  - **Signal Integrity Optimization**: Preventing issues such as crosstalk and electromagnetic interference in wire routing [299,300].
  - **Area Optimization**: Minimizing the total area occupied by the physical layout of the components [301].
- **ASIC Design**
  - **Standard Cell Library Selection**: Choosing the right standard cell libraries for performance, power, and area trade-offs [302].
  - **Custom Cell Design**: Design of custom logic cells optimized for specific performance and area requirements [303].
  - **Power Grid Optimization**: Design of efficient power distribution networks across ASICs [304].
  - **Design-for-Test (DFT)**: Embedding testability features into the design to ensure that the ASIC is testable after fabrication [305,306].
  - **ASIC Synthesis**: Translating high-level designs into a netlist using synthesis tools [307,308].
  - **Timing Closure for ASICs**: Ensure that the final design meets the timing constraints before manufacturing [309,310].
  - **Packaging and I/O Design**: Optimizing external interfaces and packaging for the ASIC [311].
- **Fault-Tolerant Design**
  - **Error Detection and Correction (EDAC)**: Design hardware with the ability to detect and correct errors (e.g., Error Correction Code (ECC)) [312,313].
  - **Redundancy Techniques**: Implementing redundancy at various levels (e.g., triple modular redundancy, hot spares) [163,314,315].
  - **Built-in Self-test (BIST)**: Design self-test circuits that can detect faults during run-time [316,317].
  - **Fault-tolerant FSMs**: Design of finite-state machines that can recover from faulty states [318,319].
  - **Radiation Hardening**: Designing hardware that is resilient to radiation effects, particularly for aerospace applications [320,321].
  - **Soft Error Mitigation**: Techniques for preventing or recovering transient errors caused by cosmic rays or other environmental factors [322,323].
  - **Failure Mode Analysis**: Analyze hardware to predict and mitigate potential failure modes [324,325].
- **Verification Plan Generation**
  - **Testbench Automation**: Automatically generating testbenches for verifying different parts of the hardware design [182,189].
  - **Random Test Generation**: Creating random test sequences to stress the design and catch edge cases [326].

 doi:10.20944/preprints202411.0156.v1

- **Constraint-based Verification**: Defining constraints for test generation to ensure valid input/output scenarios [327].
- **Formal Verification**: Using mathematical techniques to prove that a design meets its specifications [328,329].
- **Assertion-based Verification (ABV)**: Embedding assertions in the design to ensure it behaves as expected under all conditions [330,331].
- **Equivalence Checking**: Ensure that two versions of a design (e.g., RTL vs. synthesized) are functionally equivalent [332,333].
- **Universal Verification Methodology (UVM) and SystemVerilog**: Implement advanced verification techniques using UVM and SystemVerilog [334].

4.3. Methodological Improvements

With making significant improvements in hardware design and verification, several aspects of LLMs need to be reinforced. These enhancements would target specific challenges in hardware engineering and leverage LLMs to their full potential in assisting designers and verification teams. The key aspects that should be reinforced are as follows:

1) Domain-Specific Understanding and Contextual Knowledge: LLMs need a deeper and more precise understanding of hardware design languages, methodologies, and tools. While LLMs excel at NLP, they often lack in-depth knowledge of domain-specific languages like Verilog, VHDL, SystemVerilog, and UVM. To truly aid hardware designers, LLMs must be fine-tuned on vast datasets of HDLs, verification code, design documentation, and real-world projects. Additionally, understanding the context of a design, such as the specific requirements of a given project (e.g., low-power design, high-performance, etc.), will enable LLMs to make more relevant suggestions during both the design and verification processes.

2) Enhanced Formal Reasoning Capabilities: LLMs need to improve their formal reasoning abilities, especially for tasks such as formal verification, model checking, and constraint satisfaction, which are essential to hardware verification. Hardware design often involves proving that a design meets certain formal specifications, such as safety or liveness properties. Currently, LLMs struggle with formal logic and mathematical rigor. Enhancing their capability to handle formal methods—like understanding temporal logic, SVA, and finite state machines—would significantly improve their utility in verification tasks. This would allow LLMs to automatically generate and validate formal properties from natural language specifications, ensuring that hardware designs conform to their intended behavior.

3) Code Generation for Synthesis-Ready HDL: While LLMs can generate HDL code from behavioral descriptions, they must become more adept at creating synthesis-ready code. This requires not only understanding how to describe hardware behavior but also generating optimized code that meets the constraints of modern hardware synthesis tools. To achieve this, LLMs need reinforcement in optimizing generated code for real-world constraints such as timing, power, and area. Incorporating feedback from synthesis and place-and-route tools into the LLM's training data can improve its ability to generate resource-efficient, high-performance HDL designs.

4) Design Space Exploration and Optimization: One of the critical tasks in hardware design is balancing multiple design constraints—performance, power, area, and cost—through DSE. LLMs should be reinforced with advanced optimization techniques and predictive modeling capabilities to help guide the exploration of various design parameters. Reinforcement learning approaches combined with LLMs can enable them to predict the impact of parameter choices on design metrics and suggest optimal configurations based on trade-offs. By enhancing LLMs' ability to navigate complex design spaces, designers could receive better support in exploring Pareto-optimal designs that balance competing objectives.

5) Error Detection and Debugging: LLMs can play a crucial role in identifying bugs and design flaws, but their error detection capabilities must be reinforced to be more effective in the hardware domain. This includes being able to recognize subtle errors in HDL, such as incorrect state machine

 doi:10.20944/preprints202411.0156.v1

transitions, misaligned clock domain crossings, or resource contention. LLMs need to be trained on common hardware design errors and verification failures, improving their ability to offer precise feedback on potential issues. Additionally, LLMs should be reinforced with an understanding of simulation and synthesis reports, enabling them to trace errors back to their root causes and provide actionable debugging suggestions.

6) Verification Automation and Coverage Analysis: Verification is one of the most time-consuming aspects of hardware development. To improve LLMs' contributions in this domain, they should be reinforced with better tools for generating comprehensive testbenches, performing functional coverage analysis, and creating directed random tests. Specifically, LLMs should be enhanced to recognize coverage gaps in verification plans and generate appropriate tests to fill those gaps, ensuring that designs are thoroughly tested. Furthermore, improving the LLM's ability to integrate with simulation tools, extract meaningful insights from waveform data, and recommend additional verification steps will reduce the manual burden on verification teams.

7) Learning from Hardware Development Iterations: LLMs should be able to learn from previous iterations of a hardware design to assist in continuous improvement. By analyzing successive versions of a design, LLMs can identify what changes led to better performance, lower power consumption, or reduced area. Reinforcing this ability would enable LLMs to provide context-specific recommendations based on past design choices, helping hardware designers optimize their designs more effectively across multiple development cycles. This capability could also be extended to learning from community-wide datasets of hardware designs to suggest best practices and design patterns that are tailored to specific project goals.

8) Interaction with EDA Tools and Integration into Workflows: For LLMs to be more effective in hardware design, they need tighter integration with EDA tools and workflows. LLMs should be able to interface with common hardware design tools (such as simulation, synthesis, and formal verification tools), extract relevant data, and act on that information in real-time. By integrating LLMs with these tools, designers can receive real-time feedback on design choices, simulation results, or synthesis reports. LLMs should also be capable of automating repetitive tasks within the EDA workflow, such as setting up project configurations, running simulations, and analyzing results, reducing the overall design time.

9) Memory and State Tracking for Large-Scale Projects: Hardware design projects can span months or years and involve numerous changes over time. LLMs should be reinforced with better long-term memory and state-tracking capabilities so that they can keep track of ongoing changes across large projects. This would allow LLMs to assist designers by recalling relevant design decisions, tracking the evolution of specific modules or components, and ensuring consistency across the entire design. This state-tracking ability is crucial for handling complex projects with multiple designers, where coordination and memory of past decisions are key to success.

10) Security and Safety in Hardware Design: LLMs should be enhanced to understand and enforce security and safety requirements during the design process. With the growing need for hardware security, LLMs must be able to detect potential vulnerabilities, such as insecure communication protocols or improper handling of sensitive data. Similarly, in safety-critical designs, such as automotive or aerospace systems, LLMs need reinforcement to ensure compliance with safety standards and protocols. By improving LLMs' capabilities in these areas, designers can be alerted to potential security risks and safety violations early in the design phase.

5. Conclusions

This survey has explored the emerging role of LLMs in hardware design and verification, presenting an overview of the current state of the literature, key challenges, and open issues. The integration of LLMs into this domain offers significant potential to revolutionize traditional workflows, enhancing productivity and enabling more automated and intelligent design processes. From facilitating high-level hardware description generation to improving verification through natural

 doi:10.20944/preprints202411.0156.v1

language interfaces, LLMs provide promising avenues to reduce complexity, improve accuracy, and accelerate time-to-market in hardware development cycles. However, despite the promising advancements, several challenges remain. The specialized nature of hardware design and verification demands precise and domain-specific capabilities from LLMs, which are not always easy to align with the general-purpose nature of current models. Additionally, issues like explainability, the handling of large-scale designs, the ability to generate verifiably correct hardware descriptions, and integration into existing workflows highlight the limitations of current LLM-based approaches. Moreover, concerns around model reliability, data privacy, and security in industrial settings require further attention before widespread adoption.

5.1. Summary of Findings

This paper discusses the transformative role of LLMs in hardware design and verification. It highlights several key areas where LLMs are improving efficiency and accuracy in hardware design by automating repetitive tasks, generating code, and facilitating better communication among engineers. The paper emphasizes the following contributions:

1) Core Applications: LLMs are applied to various tasks, including generating HDL code, optimizing design parameters, and automating verification processes like test case generation and bug detection. They can also enhance documentation and project management.

2) Challenges: Despite the advancements, challenges such as data scarcity, the need for specialized training, and integration with existing tools remain. The complexity of hardware design requires fine-tuning LLMs for specific tasks.

3) Future Directions: The paper suggests potential areas for future research, including improving LLM integration in hardware design workflows, refining LLM-generated outputs, and addressing open issues such as handling high-dimensional data and design complexity.

In summary, LLMs hold significant potential to transform hardware design and verification by automating complex tasks, enhancing productivity, and ensuring higher quality designs. However, there are still challenges related to training, data availability, and tool integration that need to be addressed to fully realize this potential.

5.2. Implications and Recommendations

There are several key areas where future research can focus to advance the application of LLMs in hardware design and verification:

1) Domain-Specific LLMs: Developing LLMs tailored to the specific needs of hardware design and verification could enhance their effectiveness. This includes models trained on HDL, circuit layouts, and specialized verification protocols.

2) Improving Verification Capabilities: Expanding the capacity of LLMs to automatically verify hardware designs through formal methods and simulation could reduce the burden of manual verification and lead to more robust, error-free hardware.

3)Hybrid Systems: Combining LLMs with other AI and traditional formal verification techniques could result in hybrid systems that leverage the strengths of both approaches, improving the accuracy and reliability of hardware designs.

4) Explainability and Interpretability: Ensuring that LLM-generated hardware descriptions are transparent and interpretable by engineers is critical. Future research could focus on developing methods to make the reasoning behind LLM outputs more understandable and trustworthy.

5) Real-World Applications: More real-world case studies are needed to evaluate the practical utility of LLMs in large-scale hardware projects. This will provide insights into the models' performance in complex, industrial settings and help identify further areas of improvement.

6) Data Privacy and Security: Addressing concerns around the secure use of LLMs in proprietary hardware design environments, including techniques for ensuring that sensitive data remains protected during model training and deployment, will be crucial for industrial adoption.

 doi:10.20944/preprints202411.0156.v1

Author Contributions: Conceptualization, M.A.; methodology, M.A. and F.Y.; software, M.A., F.Y. and M.B.; validation, M.A., F.Y. and M.B.; formal analysis, M.A. and F.Y.; investigation, M.A. and F.Y.; resources, M.A. and F.Y.; data curation, M.A. and F.Y.; writing—original draft preparation, M.A., F.Y. and M.B.; writing—review and editing, M.A., F.Y., M.B. and A.B; visualization, M.A. and F.Y.; supervision, M.A.; project administration, M.A.; funding acquisition, A.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Pinker, S. *The language instinct: How the mind creates language*; Penguin uK, 2003.
2. Hauser, M.D.; Chomsky, N.; Fitch, W.T. The faculty of language: what is it, who has it, and how did it evolve? *science* **2002**, *298*, 1569–1579.
3. Turing, A.M. *Computing machinery and intelligence*; Springer, 2009.
4. Chernyavskiy, A.; Ilvovsky, D.; Nakov, P. Transformers:"the end of history" for natural language processing? Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part III 21. Springer, 2021, pp. 677–693.
5. Zhao, W.X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; others. A survey of large language models. *arXiv preprint arXiv:2303.18223* **2023**.
6. Bommasani, R.; Hudson, D.A.; Adeli, E.; Altman, R.; Arora, S.; von Arx, S.; Bernstein, M.S.; Bohg, J.; Bosselut, A.; Brunskill, E.; others. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258* **2021**.
7. Wei, J.; Tay, Y.; Bommasani, R.; Raffel, C.; Zoph, B.; Borgeaud, S.; Yogatama, D.; Bosma, M.; Zhou, D.; Metzler, D.; others. Emergent abilities of large language models. *arXiv preprint arXiv:2206.07682* **2022**.
8. Bahl, L.R.; Brown, P.F.; De Souza, P.V.; Mercer, R.L. A tree-based statistical language model for natural language speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing* **1989**, *37*, 1001–1008.
9. Frederick, J. Statistical methods for speech recognition, 1999.
10. Gao, J.; Lin, C.Y. Introduction to the special issue on statistical language modeling, 2004.
11. Bellegarda, J.R. Statistical language model adaptation: review and perspectives. *Speech communication* **2004**, *42*, 93–108.
12. Zhai, C.; others. Statistical language models for information retrieval a critical review. *Foundations and Trends® in Information Retrieval* **2008**, *2*, 137–213.
13. Bengio, Y.; Ducharme, R.; Vincent, P. A neural probabilistic language model. *Advances in neural information processing systems* **2000**, *13*.
14. Mikolov, T.; Karafiát, M.; Burget, L.; Cernockỳ, J.; Khudanpur, S. Recurrent neural network based language model. Interspeech. Makuhari, 2010, Vol. 2, pp. 1045–1048.
15. Kombrink, S.; Mikolov, T.; Karafiát, M.; Burget, L. Recurrent Neural Network Based Language Modeling in Meeting Recognition. Interspeech, 2011, Vol. 11, pp. 2877–2880.
16. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. *Advances in neural information processing systems* **2017**, *30*.
17. Shaw, P.; Uszkoreit, J.; Vaswani, A. Self-attention with relative position representations. *arXiv preprint arXiv:1803.02155* **2018**.
18. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805* **2018**.
19. Ghojogh, B.; Ghodsi, A. Attention mechanism, transformers, BERT, and GPT: tutorial and survey **2020**.
20. Liu, Q.; Kusner, M.J.; Blunsom, P. A survey on contextual embeddings. *arXiv preprint arXiv:2003.07278* **2020**.
21. Ge, Y.; Hua, W.; Mei, K.; Tan, J.; Xu, S.; Li, Z.; Zhang, Y.; others. Openagi: When llm meets domain experts. *Advances in Neural Information Processing Systems* **2024**, *36*.
22. Alex, N.; Lifland, E.; Tunstall, L.; Thakur, A.; Maham, P.; Riedel, C.J.; Hine, E.; Ashurst, C.; Sedille, P.; Carlier, A.; others. RAFT: A real-world few-shot text classification benchmark. *arXiv preprint arXiv:2109.14076* **2021**.

 doi:10.20944/preprints202411.0156.v1

23. Qin, C.; Zhang, A.; Zhang, Z.; Chen, J.; Yasunaga, M.; Yang, D. Is ChatGPT a general-purpose natural language processing task solver? *arXiv preprint arXiv:2302.06476* **2023**.
24. Gao, J.; Zhao, H.; Yu, C.; Xu, R. Exploring the feasibility of chatgpt for event extraction. *arXiv preprint arXiv:2303.03836* **2023**.
25. Ma, Y.; Cao, Y.; Hong, Y.; Sun, A. Large language model is not a good few-shot information extractor, but a good reranker for hard samples! *arXiv preprint arXiv:2303.08559* **2023**.
26. Cheng, D.; Huang, S.; Bi, J.; Zhan, Y.; Liu, J.; Wang, Y.; Sun, H.; Wei, F.; Deng, D.; Zhang, Q. Uprise: Universal prompt retrieval for improving zero-shot evaluation. *arXiv preprint arXiv:2303.08518* **2023**.
27. Ren, R.; Qu, Y.; Liu, J.; Zhao, W.X.; She, Q.; Wu, H.; Wang, H.; Wen, J.R. Rocketqav2: A joint training method for dense passage retrieval and passage re-ranking. *arXiv preprint arXiv:2110.07367* **2021**.
28. Sun, W.; Yan, L.; Ma, X.; Wang, S.; Ren, P.; Chen, Z.; Yin, D.; Ren, Z. Is ChatGPT good at search? investigating large language models as re-ranking agents. *arXiv preprint arXiv:2304.09542* **2023**.
29. Ziems, N.; Yu, W.; Zhang, Z.; Jiang, M. Large language models are built-in autoregressive search engines. *arXiv preprint arXiv:2305.09612* **2023**.
30. Tay, Y.; Tran, V.; Dehghani, M.; Ni, J.; Bahri, D.; Mehta, H.; Qin, Z.; Hui, K.; Zhao, Z.; Gupta, J.; others. Transformer memory as a differentiable search index. *Advances in Neural Information Processing Systems* **2022**, *35*, 21831–21843.
31. Dai, S.; Shao, N.; Zhao, H.; Yu, W.; Si, Z.; Xu, C.; Sun, Z.; Zhang, X.; Xu, J. Uncovering chatgpt's capabilities in recommender systems. Proceedings of the 17th ACM Conference on Recommender Systems, 2023, pp. 1126–1132.
32. Zheng, B.; Hou, Y.; Lu, H.; Chen, Y.; Zhao, W.X.; Wen, J.R. Adapting large language models by integrating collaborative semantics for recommendation. *arXiv preprint arXiv:2311.09049* **2023**.
33. Wang, L.; Ma, C.; Feng, X.; Zhang, Z.; Yang, H.; Zhang, J.; Chen, Z.; Tang, J.; Chen, X.; Lin, Y.; others. A survey on large language model based autonomous agents. *Frontiers of Computer Science* **2024**, *18*, 186345.
34. Wang, L.; Zhang, J.; Chen, X.; Lin, Y.; Song, R.; Zhao, W.X.; Wen, J.R. Recagent: A novel simulation paradigm for recommender systems. *arXiv preprint arXiv:2306.02552* **2023**.
35. Du, Y.; Liu, Z.; Li, J.; Zhao, W.X. A survey of vision-language pre-trained models. *arXiv preprint arXiv:2202.10936* **2022**.
36. Gan, Z.; Li, L.; Li, C.; Wang, L.; Liu, Z.; Gao, J.; others. Vision-language pre-training: Basics, recent advances, and future trends. *Foundations and Trends® in Computer Graphics and Vision* **2022**, *14*, 163–352.
37. Chen, W.; Su, Y.; Yan, X.; Wang, W.Y. KGPT: Knowledge-grounded pre-training for data-to-text generation. *arXiv preprint arXiv:2010.02307* **2020**.
38. Wang, X.; Wang, Z.; Liu, J.; Chen, Y.; Yuan, L.; Peng, H.; Ji, H. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. *arXiv preprint arXiv:2309.10691* **2023**.
39. Zhang, X.; Yu, B.; Yu, H.; Lv, Y.; Liu, T.; Huang, F.; Xu, H.; Li, Y. Wider and deeper llm networks are fairer llm evaluators. *arXiv preprint arXiv:2308.01862* **2023**.
40. Singhal, K.; Azizi, S.; Tu, T.; Mahdavi, S.S.; Wei, J.; Chung, H.W.; Scales, N.; Tanwani, A.; Cole-Lewis, H.; Pfohl, S.; others. Large language models encode clinical knowledge. *Nature* **2023**, *620*, 172–180.
41. Jeblick, K.; Schachtner, B.; Dexl, J.; Mittermeier, A.; Stüber, A.T.; Topalis, J.; Weber, T.; Wesp, P.; Sabel, B.O.; Ricke, J.; others. ChatGPT makes medicine easy to swallow: an exploratory case study on simplified radiology reports. *European radiology* **2024**, *34*, 2817–2825.
42. Chen, S.; Kann, B.H.; Foote, M.B.; Aerts, H.J.; Savova, G.K.; Mak, R.H.; Bitterman, D.S. The utility of chatgpt for cancer treatment information. medRxiv. *Preprint posted March* **2023**, *16*.
43. Singhal, K.; Tu, T.; Gottweis, J.; Sayres, R.; Wulczyn, E.; Hou, L.; Clark, K.; Pfohl, S.; Cole-Lewis, H.; Neal, D.; others. Towards expert-level medical question answering with large language models. *arXiv preprint arXiv:2305.09617* **2023**.
44. Yang, K.; Ji, S.; Zhang, T.; Xie, Q.; Ananiadou, S. On the evaluations of chatgpt and emotion-enhanced prompting for mental health analysis. *arXiv preprint arXiv:2304.03347* **2023**, *4*.
45. Tang, R.; Han, X.; Jiang, X.; Hu, X. Does synthetic data generation of llms help clinical text mining? *arXiv preprint arXiv:2303.04360* **2023**.
46. Rane, N.L.; Tawde, A.; Choudhary, S.P.; Rane, J. Contribution and performance of ChatGPT and other Large Language Models (LLM) for scientific and research advancements: a double-edged sword. *International Research Journal of Modernization in Engineering Technology and Science* **2023**, *5*, 875–899.

47. Dai, W.; Lin, J.; Jin, H.; Li, T.; Tsai, Y.S.; Gašević, D.; Chen, G. Can large language models provide feedback to students? A case study on ChatGPT. 2023 IEEE International Conference on Advanced Learning Technologies (ICALT). IEEE, 2023, pp. 323–325.
48. Young, J.C.; Shishido, M. Investigating OpenAI's ChatGPT potentials in generating Chatbot's dialogue for English as a foreign language learning. *International journal of advanced computer science and applications* **2023**, *14*.
49. Kasneci, E.; Seßler, K.; Küchemann, S.; Bannert, M.; Dementieva, D.; Fischer, F.; Gasser, U.; Groh, G.; Günnemann, S.; Hüllermeier, E.; others. ChatGPT for good? On opportunities and challenges of large language models for education. *Learning and individual differences* **2023**, *103*, 102274.
50. Susnjak, T.; McIntosh, T.R. ChatGPT: The end of online exam integrity? *Education Sciences* **2024**, *14*, 656.
51. Tamkin, A.; Brundage, M.; Clark, J.; Ganguli, D. Understanding the capabilities, limitations, and societal impact of large language models. *arXiv preprint arXiv:2102.02503* **2021**.
52. Nay, J.J. Law informs code: A legal informatics approach to aligning artificial intelligence with humans. *Nw. J. Tech. & Intell. Prop.* **2022**, *20*, 309.
53. Yu, F.; Quartey, L.; Schilder, F. Legal prompting: Teaching a language model to think like a lawyer. *arXiv preprint arXiv:2212.01326* **2022**.
54. Trautmann, D.; Petrova, A.; Schilder, F. Legal prompt engineering for multilingual legal judgement prediction. *arXiv preprint arXiv:2212.02199* **2022**.
55. Sun, Z. A short survey of viewing large language models in legal aspect. *arXiv preprint arXiv:2303.09136* **2023**.
56. Savelka, J.; Ashley, K.D.; Gray, M.A.; Westermann, H.; Xu, H. Explaining legal concepts with augmented large language models (gpt-4). *arXiv preprint arXiv:2306.09525* **2023**.
57. Cui, J.; Li, Z.; Yan, Y.; Chen, B.; Yuan, L. Chatlaw: Open-source legal large language model with integrated external knowledge bases. *arXiv preprint arXiv:2306.16092* **2023**.
58. Guha, N.; Nyarko, J.; Ho, D.; Ré, C.; Chilton, A.; Chohlas-Wood, A.; Peters, A.; Waldon, B.; Rockmore, D.; Zambrano, D.; others. Legalbench: A collaboratively built benchmark for measuring legal reasoning in large language models. *Advances in Neural Information Processing Systems* **2024**, *36*.
59. Araci, D. Finbert: Financial sentiment analysis with pre-trained language models. *arXiv preprint arXiv:1908.10063* **2019**.
60. Li, Y.; Wang, S.; Ding, H.; Chen, H. Large language models in finance: A survey. Proceedings of the fourth ACM international conference on AI in finance, 2023, pp. 374–382.
61. Yang, H.; Liu, X.Y.; Wang, C.D. Fingpt: Open-source financial large language models. *arXiv preprint arXiv:2306.06031* **2023**.
62. Son, G.; Jung, H.; Hahm, M.; Na, K.; Jin, S. Beyond classification: Financial reasoning in state-of-the-art language models. *arXiv preprint arXiv:2305.01505* **2023**.
63. Shah, A.; Chava, S. Zero is not hero yet: Benchmarking zero-shot performance of llms for financial tasks. *arXiv preprint arXiv:2305.16633* **2023**.
64. Jin, Q.; Dhingra, B.; Liu, Z.; Cohen, W.W.; Lu, X. Pubmedqa: A dataset for biomedical research question answering. *arXiv preprint arXiv:1909.06146* **2019**.
65. Mahadi Hassan, M.; Knipper, A.; Kanti Karmaker Santu, S. ChatGPT as your Personal Data Scientist. *arXiv e-prints* **2023**, pp. arXiv–2305.
66. Irons, J.; Mason, C.; Cooper, P.; Sidra, S.; Reeson, A.; Paris, C. Exploring the Impacts of ChatGPT on Future Scientific Work **2023**.
67. Altmäe, S.; Sola-Leyva, A.; Salumets, A. Artificial intelligence in scientific writing: a friend or a foe? *Reproductive BioMedicine Online* **2023**, *47*, 3–9.
68. Zheng, Y.; Koh, H.Y.; Ju, J.; Nguyen, A.T.; May, L.T.; Webb, G.I.; Pan, S. Large language models for scientific synthesis, inference and explanation. *arXiv preprint arXiv:2310.07984* **2023**.
69. Aczel, B.; Wagenmakers, E.J. Transparency guidance for ChatGPT usage in scientific writing **2023**.
70. Jin, H.; Huang, L.; Cai, H.; Yan, J.; Li, B.; Chen, H. From llms to llm-based agents for software engineering: A survey of current, challenges and future. *arXiv preprint arXiv:2408.02479* **2024**.
71. Kimura, A.; Scholl, J.; Schaffranek, J.; Sutter, M.; Elliott, A.; Strizich, M.; Via, G.D. A decomposition workflow for integrated circuit verification and validation. *Journal of Hardware and Systems Security* **2020**, *4*, 34–43.

 doi:10.20944/preprints202411.0156.v1

72. Roy, D.; Zhang, X.; Bhave, R.; Bansal, C.; Las-Casas, P.; Fonseca, R.; Rajmohan, S. Exploring llm-based agents for root cause analysis. Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering, 2024, pp. 208–219.
73. Guo, C.; Cheng, F.; Du, Z.; Kiessling, J.; Ku, J.; Li, S.; Li, Z.; Ma, M.; Molom-Ochir, T.; Morris, B.; others. A Survey: Collaborative Hardware and Software Design in the Era of Large Language Models. *arXiv preprint arXiv:2410.07265* **2024**.
74. Xu, N.; Zhang, Z.; Qi, L.; Wang, W.; Zhang, C.; Ren, Z.; Zhang, H.; Cheng, X.; Zhang, Y.; Liu, Z.; others. ChipExpert: The Open-Source Integrated-Circuit-Design-Specific Large Language Model. *arXiv preprint arXiv:2408.00804* **2024**.
75. Zheng, Y.; Chen, Y.; Qian, B.; Shi, X.; Shu, Y.; Chen, J. A Review on Edge Large Language Models: Design, Execution, and Applications. *arXiv preprint arXiv:2410.11845* **2024**.
76. Hirschberg, J.; Ballard, B.W.; Hindle, D. Natural language processing. *AT&T technical journal* **1988**, *67*, 41–57.
77. Petrushin, V.A. Hidden markov models: Fundamentals and applications. Online Symposium for Electronics Engineer, 2000.
78. Yin, W.; Kann, K.; Yu, M.; Schütze, H. Comparative study of CNN and RNN for natural language processing. *arXiv preprint arXiv:1702.01923* **2017**.
79. Hihi, S.; Bengio, Y. Hierarchical recurrent neural networks for long-term dependencies. *Advances in neural information processing systems* **1995**, *8*.
80. Hochreiter, S. Recurrent neural net learning and vanishing gradient. *International Journal Of Uncertainity, Fuzziness and Knowledge-Based Systems* **1998**, *6*, 107–116.
81. Azunre, P. *Transfer learning for natural language processing*; Simon and Schuster, 2021.
82. Shi, Y.; Larson, M.; Jonker, C.M. Recurrent neural network language model adaptation with curriculum learning. *Computer Speech & Language* **2015**, *33*, 136–154.
83. Kovačević, A.; Kečo, D. Bidirectional LSTM networks for abstractive text summarization. Advanced Technologies, Systems, and Applications VI: Proceedings of the International Symposium on Innovative and Interdisciplinary Applications of Advanced Technologies (IAT) 2021. Springer, 2022, pp. 281–293.
84. Wu, Y.; Schuster, M.; Chen, Z.; Le, Q.V.; Norouzi, M.; Macherey, W.; Krikun, M.; Cao, Y.; Gao, Q.; Macherey, K.; others. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144* **2016**.
85. Yadav, R.K.; Harwani, S.; Maurya, S.K.; Kumar, S. Intelligent Chatbot Using GNMT, SEQ-2-SEQ Techniques. 2021 International Conference on Intelligent Technologies (CONIT). IEEE, 2021, pp. 1–5.
86. Luitse, D.; Denkena, W. The great transformer: Examining the role of large language models in the political economy of AI. *Big Data & Society* **2021**, *8*, 20539517211047734.
87. Topal, M.O.; Bas, A.; van Heerden, I. Exploring transformers in natural language generation: Gpt, bert, and xlnet. *arXiv preprint arXiv:2102.08036* **2021**.
88. Bird, J.J.; Ekárt, A.; Faria, D.R. Chatbot Interaction with Artificial Intelligence: human data augmentation with T5 and language transformer ensemble for text classification. *Journal of Ambient Intelligence and Humanized Computing* **2023**, *14*, 3129–3144.
89. Radford, A.; Narasimhan, K.; Salimans, T.; Sutskever, I.; others. Improving language understanding by generative pre-training **2018**.
90. Radford, A.; Wu, J.; Amodei, D.; Amodei, D.; Clark, J.; Brundage, M.; Sutskever, I. Better language models and their implications. *OpenAI blog* **2019**, *1*.
91. Brown, T.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J.D.; Dhariwal, P.; Neelakantan, A.; Shyam, P.; Sastry, G.; Askell, A.; others. Language models are few-shot learners. *Advances in neural information processing systems* **2020**, *33*, 1877–1901.
92. Achiam, J.; Adler, S.; Agarwal, S.; Ahmad, L.; Akkaya, I.; Aleman, F.L.; Almeida, D.; Altenschmidt, J.; Altman, S.; Anadkat, S.; others. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774* **2023**.
93. Huang, J.; Chang, K.C.C. Towards reasoning in large language models: A survey. *arXiv preprint arXiv:2212.10403* **2022**.
94. Xi, Z.; Chen, W.; Guo, X.; He, W.; Ding, Y.; Hong, B.; Zhang, M.; Wang, J.; Jin, S.; Zhou, E.; others. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864* **2023**.

95. Hadi, M.U.; Al Tashi, Q.; Shah, A.; Qureshi, R.; Muneer, A.; Irfan, M.; Zafar, A.; Shaikh, M.B.; Akhtar, N.; Wu, J.; others. Large language models: a comprehensive survey of its applications, challenges, limitations, and future prospects. *Authorea Preprints* **2024**.
96. Naveed, H.; Khan, A.U.; Qiu, S.; Saqib, M.; Anwar, S.; Usman, M.; Barnes, N.; Mian, A. A comprehensive overview of large language models. *arXiv preprint arXiv:2307.06435* **2023**.
97. Fan, L.; Li, L.; Ma, Z.; Lee, S.; Yu, H.; Hemphill, L. A bibliometric review of large language models research from 2017 to 2023. *arXiv preprint arXiv:2304.02020* **2023**.
98. Raiaan, M.A.K.; Mukta, M.S.H.; Fatema, K.; Fahad, N.M.; Sakib, S.; Mim, M.M.J.; Ahmad, J.; Ali, M.E.; Azam, S. A review on large Language Models: Architectures, applications, taxonomies, open issues and challenges. *IEEE Access* **2024**.
99. Minaee, S.; Mikolov, T.; Nikzad, N.; Chenaghlu, M.; Socher, R.; Amatriain, X.; Gao, J. Large language models: A survey. *arXiv preprint arXiv:2402.06196* **2024**.
100. Liu, Y.; He, H.; Han, T.; Zhang, X.; Liu, M.; Tian, J.; Zhang, Y.; Wang, J.; Gao, X.; Zhong, T.; others. Understanding llms: A comprehensive overview from training to inference. *arXiv preprint arXiv:2401.02038* **2024**.
101. Cui, C.; Ma, Y.; Cao, X.; Ye, W.; Zhou, Y.; Liang, K.; Chen, J.; Lu, J.; Yang, Z.; Liao, K.D.; others. A survey on multimodal large language models for autonomous driving. Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, 2024, pp. 958–979.
102. Chang, Y.; Wang, X.; Wang, J.; Wu, Y.; Yang, L.; Zhu, K.; Chen, H.; Yi, X.; Wang, C.; Wang, Y.; others. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* **2024**, *15*, 1–45.
103. Kachris, C. A survey on hardware accelerators for large language models. *arXiv preprint arXiv:2401.09890* **2024**.
104. Zhang, H.; Ning, A.; Prabhakar, R.; Wentzlaff, D. A Hardware Evaluation Framework for Large Language Model Inference. *arXiv preprint arXiv:2312.03134* **2023**.
105. Korvala, A. Analysis of LLM-models in optimizing and designing VHDL code. Master's thesis, Modern SW and Computing technolgies, Oulu University of Applied Sciences, 2023.
106. Thakur, S.; Blocklove, J.; Pearce, H.; Tan, B.; Garg, S.; Karri, R. Autochip: Automating hdl generation using llm feedback. *arXiv preprint arXiv:2311.04887* **2023**.
107. Thakur, S.; Ahmad, B.; Fan, Z.; Pearce, H.; Tan, B.; Karri, R.; Dolan-Gavitt, B.; Garg, S. Benchmarking large language models for automated verilog rtl code generation. In 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE), 2023.
108. Blocklove, J.; Garg, S.; Karri, R.; Pearce, H. Chip-chat: Challenges and opportunities in conversational hardware design. 2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD). IEEE, 2023, pp. 1–6.
109. Chang, K.; Wang, Y.; Ren, H.; Wang, M.; Liang, S.; Han, Y.; Li, H.; Li, X. Chipgpt: How far are we from natural language hardware design. *arXiv preprint arXiv:2305.14019* **2023**.
110. Martínez, P.A.; Bernabé, G.; García, J.M. Code Detection for Hardware Acceleration Using Large Language Models. *IEEE Access* **2024**.
111. DeLorenzo, M.; Gohil, V.; Rajendran, J. CreativEval: Evaluating Creativity of LLM-Based Hardware Code Generation. *arXiv preprint arXiv:2404.08806* **2024**.
112. Tomlinson, M.; Li, J.; Andreou, A. Designing Silicon Brains using LLM: Leveraging ChatGPT for Automated Description of a Spiking Neuron Array. *arXiv preprint arXiv:2402.10920* **2024**.
113. Xiang, M.; Goh, E.; Teo, T.H. Digital ASIC Design with Ongoing LLMs: Strategies and Prospects. *arXiv preprint arXiv:2405.02329* **2024**.
114. Wang, H.; others. Efficient algorithms and hardware for natural language processing. PhD thesis, Massachusetts Institute of Technology, 2020.
115. Fu, Y.; Zhang, Y.; Yu, Z.; Li, S.; Ye, Z.; Li, C.; Wan, C.; Lin, Y.C. Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models. 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD). IEEE, 2023, pp. 1–9.
116. Fu, W.; Li, S.; Zhao, Y.; Ma, H.; Dutta, R.; Zhang, X.; Yang, K.; Jin, Y.; Guo, X. Hardware Phi-1.5 B: A Large Language Model Encodes Hardware Domain Specific Knowledge. *arXiv preprint arXiv:2402.01728* **2024**.

 doi:10.20944/preprints202411.0156.v1

117. Wang, H.; Wu, Z.; Liu, Z.; Cai, H.; Zhu, L.; Gan, C.; Han, S. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187* **2020**.
118. Chang, K.; Ren, H.; Wang, M.; Liang, S.; Han, Y.; Li, H.; Li, X.; Wang, Y. Improving Large Language Model Hardware Generating Quality through Post-LLM Search. Machine Learning for Systems 2023, 2023.
119. Guo, C.; Tang, J.; Hu, W.; Leng, J.; Zhang, C.; Yang, F.; Liu, Y.; Guo, M.; Zhu, Y. Olive: Accelerating large language models via hardware-friendly outlier-victim pair quantization. Proceedings of the 50th Annual International Symposium on Computer Architecture, 2023, pp. 1–15.
120. Liu, S.; Fang, W.; Lu, Y.; Zhang, Q.; Zhang, H.; Xie, Z. Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution. *arXiv preprint arXiv:2312.08617* **2023**.
121. Lu, Y.; Liu, S.; Zhang, Q.; Xie, Z. Rtllm: An open-source benchmark for design rtl generation with large language model. 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2024, pp. 722–727.
122. Pandelea, V.; Ragusa, E.; Gastaldo, P.; Cambria, E. Selecting Language Models Features VIA Software-Hardware Co-Design. ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, 2023, pp. 1–5.
123. Cerisara, C. SlowLLM: large language models on consumer hardware. PhD thesis, CNRS, 2023.
124. Li, M.; Fang, W.; Zhang, Q.; Xie, Z. Specllm: Exploring generation and review of vlsi design specification with large language model. *arXiv preprint arXiv:2401.13266* **2024**.
125. Kurtić, E.; Frantar, E.; Alistarh, D. ZipLM: Inference-Aware Structured Pruning of Language Models. *Advances in Neural Information Processing Systems* **2024**, *36*.
126. Thorat, K.; Zhao, J.; Liu, Y.; Peng, H.; Xie, X.; Lei, B.; Zhang, J.; Ding, C. Advanced language model-driven verilog development: Enhancing power, performance, and area optimization in code synthesis. *arXiv preprint arXiv:2312.01022* **2023**.
127. Huang, Y.; Wan, L.J.; Ye, H.; Jha, M.; Wang, J.; Li, Y.; Zhang, X.; Chen, D. New Solutions on LLM Acceleration, Optimization, and Application. *arXiv preprint arXiv:2406.10903* **2024**.
128. Goh, E.; Xiang, M.; Wey, I.; Teo, T.H.; others. From English to ASIC: Hardware Implementation with Large Language Model. *arXiv preprint arXiv:2403.07039* **2024**.
129. Mudigere, D.; Hao, Y.; Huang, J.; Jia, Z.; Tulloch, A.; Sridharan, S.; Liu, X.; Ozdal, M.; Nie, J.; Park, J.; others. Software-hardware co-design for fast and scalable training of deep learning recommendation models. Proceedings of the 49th Annual International Symposium on Computer Architecture, 2022, pp. 993–1011.
130. Wan, L.J.; Huang, Y.; Li, Y.; Ye, H.; Wang, J.; Zhang, X.; Chen, D. Software/Hardware Co-design for LLM and Its Application for Design Verification. 2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC). IEEE, 2024, pp. 435–441.
131. Yan, Z.; Qin, Y.; Hu, X.S.; Shi, Y. On the viability of using llms for sw/hw co-design: An example in designing cim dnn accelerators. 2023 IEEE 36th International System-on-Chip Conference (SOCC). IEEE, 2023, pp. 1–6.
132. Nazzal, M.; Vungarala, D.; Morsali, M.; Zhang, C.; Ghosh, A.; Khreishah, A.; Angizi, S. A Dataset for Large Language Model-Driven AI Accelerator Generation. *arXiv preprint arXiv:2404.10875* **2024**.
133. Vungarala, D.L.V.D. Gen-acceleration: Pioneering work for hardware accelerator generation using large language models. Master's thesis, Electrical and Computer Engineering, New Jersey Institute of Technology, 2023.
134. Heo, G.; Lee, S.; Cho, J.; Choi, H.; Lee, S.; Ham, H.; Kim, G.; Mahajan, D.; Park, J. NeuPIMs: NPU-PIM Heterogeneous Acceleration for Batched LLM Inferencing. Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3, 2024, pp. 722–737.
135. Lai, C.; Zhou, Z.; Poptani, A.; Zhang, W. LCM: LLM-focused Hybrid SPM-cache Architecture with Cache Management for Multi-Core AI Accelerators. Proceedings of the 38th ACM International Conference on Supercomputing, 2024, pp. 62–73.
136. Paria, S.; Dasgupta, A.; Bhunia, S. Divas: An llm-based end-to-end framework for soc security analysis and policy-based protection. *arXiv preprint arXiv:2308.06932* **2023**.
137. Srikumar, P. Fast and wrong: The case for formally specifying hardware with LLMS. Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS). ACM. ACM Press, 2023.

 doi:10.20944/preprints202411.0156.v1

138. Ahmad, B.; Thakur, S.; Tan, B.; Karri, R.; Pearce, H. Fixing hardware security bugs with large language models. *arXiv preprint arXiv:2302.01215* **2023**.
139. Kokolakis, G.; Moschos, A.; Keromytis, A.D. Harnessing the power of general-purpose llms in hardware trojan design. Proceedings of the 5th Workshop on Artificial Intelligence in Hardware Security, in conjunction with ACNS, 2024, Vol. 14.
140. Saha, D.; Tarek, S.; Yahyaei, K.; Saha, S.K.; Zhou, J.; Tehranipoor, M.; Farahmandi, F. Llm for soc security: A paradigm shift. *arXiv preprint arXiv:2310.06046* **2023**.
141. Wang, Z.; Alrahis, L.; Mankali, L.; Knechtel, J.; Sinanoglu, O. LLMs and the Future of Chip Design: Unveiling Security Risks and Building Trust. *arXiv preprint arXiv:2405.07061* **2024**.
142. Ahmad, B.; Thakur, S.; Tan, B.; Karri, R.; Pearce, H. On hardware security bug code fixes by prompting large language models. *IEEE Transactions on Information Forensics and Security* **2024**.
143. Kande, R.; Pearce, H.; Tan, B.; Dolan-Gavitt, B.; Thakur, S.; Karri, R.; Rajendran, J. (Security) Assertions by Large Language Models. *IEEE Transactions on Information Forensics and Security* **2024**.
144. Tarek, S.; Saha, D.; Saha, S.K.; Tehranipoor, M.; Farahmandi, F. SoCureLLM: An LLM-driven Approach for Large-Scale System-on-Chip Security Verification and Policy Generation. *Cryptology ePrint Archive* **2024**.
145. Paria, S.; Dasgupta, A.; Bhunia, S. Navigating SoC Security Landscape on LLM-Guided Paths. Proceedings of the Great Lakes Symposium on VLSI 2024, 2024, pp. 252–257.
146. Yao, X.; Li, H.; Chan, T.H.; Xiao, W.; Yuan, M.; Huang, Y.; Chen, L.; Yu, B. Hdldebugger: Streamlining hdl debugging with large language models. *arXiv preprint arXiv:2403.11671* **2024**.
147. Fu, W.; Yang, K.; Dutta, R.G.; Guo, X.; Qu, G. LLM4SecHW: Leveraging domain-specific large language model for hardware debugging. 2023 Asian Hardware Oriented Security and Trust Symposium (AsianHOST). IEEE, 2023, pp. 1–6.
148. Fang, W.; Li, M.; Li, M.; Yan, Z.; Liu, S.; Zhang, H.; Xie, Z. AssertLLM: Generating and Evaluating Hardware Verification Assertions from Design Specifications via Multi-LLMs. *arXiv preprint arXiv:2402.00386* **2024**.
149. Orenes-Vera, M.; Martonosi, M.; Wentzlaff, D. Using llms to facilitate formal verification of rtl. *arXiv e-prints* **2023**, pp. arXiv–2309.
150. Varambally, B.S.; Sehgal, N. Optimising design verification using machine learning: An open source solution. *arXiv preprint arXiv:2012.02453* **2020**.
151. Liu, M.; Pinckney, N.; Khailany, B.; Ren, H. Verilogeval: Evaluating large language models for verilog code generation. 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD). IEEE, 2023, pp. 1–8.
152. Sun, C.; Hahn, C.; Trippel, C. Towards improving verification productivity with circuit-aware translation of natural language to systemverilog assertions. First International Workshop on Deep Learning-aided Verification, 2023.
153. Liu, M.; Ene, T.D.; Kirby, R.; Cheng, C.; Pinckney, N.; Liang, R.; Alben, J.; Anand, H.; Banerjee, S.; Bayraktaroglu, I.; others. Chipnemo: Domain-adapted llms for chip design. *arXiv preprint arXiv:2311.00176* **2023**.
154. Zhang, Z.; Chadwick, G.; McNally, H.; Zhao, Y.; Mullins, R. Llm4dv: Using large language models for hardware test stimuli generation. *arXiv preprint arXiv:2310.04535* **2023**.
155. Kande, R.; Pearce, H.; Tan, B.; Dolan-Gavitt, B.; Thakur, S.; Karri, R.; Rajendran, J. Llm-assisted generation of hardware assertions. *arXiv preprint arXiv:2306.14027* **2023**.
156. Makatura, L.; Foshey, M.; Wang, B.; Hähnlein, F.; Ma, P.; Deng, B.; Tjandrasuwita, M.; Spielberg, A.; Owens, C.E.; Chen, P.Y.; Zhao, A.; Zhu, A.; Norton, W.J.; Gu, E.; Jacob, J.; Li, Y.; Schulz, A.; Matusik, W. Large Language Models for Design and Manufacturing. *An MIT Exploration of Generative AI* **2024**. https://mit-genai.pubpub.org/pub/nmypmnhs.
157. Du, Y.; Deng, H.; Liew, S.C.; Chen, K.; Shao, Y.; Chen, H. The Power of Large Language Models for Wireless Communication System Development: A Case Study on FPGA Platforms, 2024, [arXiv:eess.SP/2307.07319].
158. Englhardt, Z.; Li, R.; Nissanka, D.; Zhang, Z.; Narayanswamy, G.; Breda, J.; Liu, X.; Patel, S.; Iyer, V. Exploring and Characterizing Large Language Models For Embedded System Development and Debugging, 2023, [arXiv:cs.SE/2307.03817].
159. Thorat, K.; Zhao, J.; Liu, Y.; Peng, H.; Xie, X.; Lei, B.; Zhang, J.; Ding, C. Advanced Large Language Model (LLM)-Driven Verilog Development: Enhancing Power, Performance, and Area Optimization in Code Synthesis, 2024, [arXiv:cs.LG/2312.01022].

160. Lian, X.; Chen, Y.; Cheng, R.; Huang, J.; Thakkar, P.; Zhang, M.; Xu, T. Configuration Validation with Large Language Models, 2024, [arXiv:cs.SE/2310.09690].
161. Sandal, S.; Akturk, I. Zero-Shot RTL Code Generation with Attention Sink Augmented Large Language Models. *arXiv preprint arXiv:2401.08683* **2024**.
162. Parchamdar, B.; Schafer, B.C. Finding Bugs in RTL Descriptions: High-Level Synthesis to the Rescue. Proceedings of 61st Design Automation Conference (DAC), 2024.
163. Tavana, M.K.; Teimouri, N.; Abdollahi, M.; Goudarzi, M. Simultaneous hardware and time redundancy with online task scheduling for low energy highly reliable standby-sparing system. *ACM Trans. Embed. Comput. Syst.* **2014**, *13*, 86–1.
164. Luo, Q.; Hu, S.; Li, C.; Li, G.; Shi, W. Resource scheduling in edge computing: A survey. *IEEE Communications Surveys & Tutorials* **2021**, *23*, 2131–2165.
165. Kumar, S.; Singh, S.K.; Aggarwal, N.; Gupta, B.B.; Alhalabi, W.; Band, S.S. An efficient hardware supported and parallelization architecture for intelligent systems to overcome speculative overheads. *International Journal of Intelligent Systems* **2022**, *37*, 11764–11790.
166. Kao, S.C.; Jeong, G.; Krishna, T. Confuciux: Autonomous hardware resource assignment for dnn accelerators using reinforcement learning. 2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO). IEEE, 2020, pp. 622–636.
167. Alwan, E.H.; Ketran, R.M.; Hussein, I.A. A Comprehensive Survey on Loop Unrolling Technique In Code Optimization. *Journal of University of Babylon for Pure and Applied Sciences* **2024**, pp. 108–117.
168. Liu, Y.; Ma, Y.; Zhang, B.; Liu, L.; Wang, J.; Tang, S. Improving the computational efficiency and flexibility of FPGA-based CNN accelerator through loop optimization. *Microelectronics Journal* **2024**, *147*, 106197.
169. Hasan, B.M.S.; Abdulazeez, A.M. A review of principal component analysis algorithm for dimensionality reduction. *Journal of Soft Computing and Data Mining* **2021**, *2*, 20–30.
170. Wang, Q.; Li, X.; Yue, C.; He, Y. A Survey of Control Flow Graph Recovery for Binary Code. CCF National Conference of Computer Applications. Springer, 2023, pp. 225–244.
171. Talati, N.; May, K.; Behroozi, A.; Yang, Y.; Kaszyk, K.; Vasiladiotis, C.; Verma, T.; Li, L.; Nguyen, B.; Sun, J.; others. Prodigy: Improving the memory latency of data-indirect irregular workloads using hardware-software co-design. 2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2021, pp. 654–667.
172. Ayers, G.; Litz, H.; Kozyrakis, C.; Ranganathan, P. Classifying memory access patterns for prefetching. Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, 2020, pp. 513–526.
173. Kastner, R.; Gong, W.; Hao, X.; Brewer, F.; Kaplan, A.; Brisk, P.; Sarrafzadeh, M. Physically Aware Data Communication Optimization for Hardware Synthesis.
174. Fan, Z. Automatically Generating Verilog RTL Code with Large Language Models. Master's thesis, New York University Tandon School of Engineering, 2023.
175. Lekidis, A. Automated Code Generation for Industrial Applications Based on Configurable Programming Models **2023**.
176. Bhandari, J.; Knechtel, J.; Narayanaswamy, R.; Garg, S.; Karri, R. LLM-Aided Testbench Generation and Bug Detection for Finite-State Machines. *arXiv preprint arXiv:2406.17132* **2024**.
177. Kibria, R.; Farahmandi, F.; Tehranipoor, M. FSMx-Ultra: Finite State Machine Extraction from Gate-Level Netlist for Security Assessment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2023**, *42*, 3613–3627.
178. Gauthier, L.; Ishikawa, Y. HDLRuby: A Ruby Extension for Hardware Description and its Translation to Synthesizable Verilog HDL. *ACM Transactions on Embedded Computing Systems* **2024**, *23*, 1–26.
179. Rashid, M.I.; Schaefer, B.C. VeriPy: A Python-Powered Framework for Parsing Verilog HDL and High-Level Behavioral Analysis of Hardware. 2024 IEEE 17th Dallas Circuits and Systems Conference (DCAS). IEEE, 2024, pp. 1–6.
180. Morgan, F.; Byrne, J.P.; Bupathi, A.; George, R.; Elahi, A.; Callaly, F.; Kelly, S.; O'Loughlin, D. HDLGen-ChatGPT Case Study: RISC-V Processor VHDL and Verilog Model-Testbench and EDA Project Generation. Proceedings of the 34th International Workshop on Rapid System Prototyping, 2023, pp. 1–7.
181. Kumar, B.; Nanda, S.; Parthasarathy, G.; Patil, P.; Tsai, A.; Choudhary, P. HDL-GPT: High-Quality HDL is All You Need. *arXiv preprint arXiv:2407.18423* **2024**.

182. Qiu, R.; Zhang, G.L.; Drechsler, R.; Schlichtmann, U.; Li, B. AutoBench: Automatic Testbench Generation and Evaluation Using LLMs for HDL Design. Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD, 2024, pp. 1–10.
183. Wenzel, J.; Hochberger, C. Automatically Restructuring HDL Modules for Improved Reusability in Rapid Synthesis. 2022 IEEE International Workshop on Rapid System Prototyping (RSP). IEEE, 2022, pp. 43–49.
184. Witharana, H.; Lyu, Y.; Charles, S.; Mishra, P. A survey on assertion-based hardware verification. *ACM Computing Surveys (CSUR)* **2022**, *54*, 1–33.
185. Agostini, N.B.; Haris, J.; Gibson, P.; Jayaweera, M.; Rubin, N.; Tumeo, A.; Abellán, J.L.; Cano, J.; Kaeli, D. AXI4MLIR: User-Driven Automatic Host Code Generation for Custom AXI-Based Accelerators. 2024 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). IEEE, 2024, pp. 143–157.
186. Vivekananda, A.A.; Enoiu, E. Automated test case generation for digital system designs: A mapping study on vhdl, verilog, and systemverilog description languages. *Designs* **2020**, *4*, 31.
187. Nuocheng, W. HDL Synthesis, Inference and Technology Mapping Algorithms for FPGA Configuration. *International Journal of Engineering and Technology* **2024**, *16*, 32–38.
188. Cardona Nadal, J. Practical strategies to monitor and control contention in shared resources of critical real-time embedded systems. PhD thesis, Universitat Politècnica de Catalunya, 2023.
189. Jayasena, A.; Mishra, P. Directed test generation for hardware validation: A survey. *ACM Computing Surveys* **2024**, *56*, 1–36.
190. Srivastava, A.; Mukherjee, R.; Marschner, E.; Seeley, C.; Dobre, S. Low Power SoC Verification: IP Reuse and Hierarchical Composition using UPF. *DVCon proceedings* **2012**.
191. Mullane, B.; MacNamee, C. Developing a reusable IP platform within a System-on-Chip design framework targeted towards an academic R&D environment. *Design and Reuse* **2008**.
192. Leipnitz, M.T.; Nazar, G.L. High-level synthesis of approximate designs under real-time constraints. *ACM Transactions on Embedded Computing Systems (TECS)* **2019**, *18*, 1–21.
193. Gangadharan, S.; Churiwala, S. *Constraining Designs for Synthesis and Timing Analysis*; Springer, 2013.
194. Namazi, A.; Abdollahi, M. PCG: Partially clock-gating approach to reduce the power consumption of fault-tolerant register files. 2017 Euromicro Conference on Digital System Design (DSD). IEEE, 2017, pp. 323–328.
195. Namazi, A.; Abdollahi, M.; Safari, S.; Mohammadi, S. LORAP: low-overhead power and reliability-aware task mapping based on instruction footprint for real-time applications. 2017 Euromicro Conference on Digital System Design (DSD). IEEE, 2017, pp. 364–367.
196. Alireza, N.; Meisam, A. LPVM: Low-Power Variation-Mitigant Adder Architecture Using Carry Expedition. Workshop on Early Reliability Modeling for Aging and Variability in Silicon Systems, 2016, pp. 41–44.
197. Chandra, A.; Chattopadhyay, S. Design of hardware efficient FIR filter: A review of the state-of-the-art approaches. *Engineering Science and Technology, an International Journal* **2016**, *19*, 212–226.
198. Narayanan, D.; Harlap, A.; Phanishayee, A.; Seshadri, V.; Devanur, N.R.; Ganger, G.R.; Gibbons, P.B.; Zaharia, M. PipeDream: Generalized pipeline parallelism for DNN training. Proceedings of the 27th ACM symposium on operating systems principles, 2019, pp. 1–15.
199. Osawa, K.; Li, S.; Hoefler, T. PipeFisher: Efficient training of large language models using pipelining and Fisher information matrices. *Proceedings of Machine Learning and Systems* **2023**, *5*, 708–727.
200. Shibo, C.; Zhang, H.; Todd, A. Zipper: Latency-Tolerant Optimizations for High-Performance Buses. To Appear in The Asia and South Pacific Design Automation Conference, 2025.
201. Shammasi, M.; Baharloo, M.; Abdollahi, M.; Baniasadi, A. Turn-aware application mapping using reinforcement learning in power gating-enabled network on chip. 2022 IEEE 15th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC). IEEE, 2022, pp. 345–352.
202. Aligholipour, R.; Baharloo, M.; Farzaneh, B.; Abdollahi, M.; Khonsari, A. TAMA: turn-aware mapping and architecture–a power-efficient network-on-chip approach. *ACM Transactions on Embedded Computing Systems (TECS)* **2021**, *20*, 1–24.
203. Jayakrishnan, M.; Chang, A.; Kim, T.T.H. Power and area efficient clock stretching and critical path reshaping for error resilience. *Journal of Low Power Electronics and Applications* **2019**, *9*, 5.
204. Smith, F.; Van den Berg, A.E. Hardware genetic algorithm optimisation by critical path analysis using a custom VLSI architecture. *South African Computer Journal* **2015**, *56*, 120–135.

 doi:10.20944/preprints202411.0156.v1

205. Barkalov, A.; Titarenko, L.; Mielcarek, K.; Mazurkiewicz, M. Hardware reduction for FSMs with extended state codes. *IEEE Access* **2024**.
206. Barkalov, A.; Titarenko, L.; CHMIELEWSKI, S. Hardware reduction in CPLD-based Moore FSM. *Journal of Circuits, Systems and Computers* **2014**, *23*, 1450086.
207. Barkalov, A.; Titarenko, L.; Malcheva, R.; Soldatov, K. Hardware reduction in FPGA-based Moore FSM. *Journal of Circuits, Systems and Computers* **2013**, *22*, 1350006.
208. Fummi, F.; Sciuto, D. A complete testing strategy based on interacting and hierarchical FSMs. *Integration* **1997**, *23*, 75–93.
209. Farahmandi, F.; Rahman, M.S.; Rajendran, S.R.; Tehranipoor, M. CAD for Fault Injection Detection. In *CAD for Hardware Security*; Springer, 2023; pp. 149–168.
210. Minns, P.D. *Digital System Design Using FSMs: A Practical Learning Approach*; John Wiley & Sons, 2021.
211. Barkalov, A.; Titarenko, L.; Bieganowski, J.; Krzywicki, K. Basic Approaches for Reducing Power Consumption in Finite State Machine Circuits—A Review. *Applied Sciences* **2024**, *14*, 2693.
212. Okada, S.; Ohzeki, M.; Taguchi, S. Efficient partition of integer optimization problems with one-hot encoding. *Scientific reports* **2019**, *9*, 13036.
213. Uyar, M.Ü.; Fecko, M.A.; Sethi, A.S.; Amer, P.D. Testing protocols modeled as FSMs with timing parameters. *Computer Networks* **1999**, *31*, 1967–1988.
214. Amir, M.; Givargis, T. Pareto optimal design space exploration of cyber-physical systems. *Internet of things* **2020**, *12*, 100308.
215. Tian, Y.; Si, L.; Zhang, X.; Cheng, R.; He, C.; Tan, K.C.; Jin, Y. Evolutionary large-scale multi-objective optimization: A survey. *ACM Computing Surveys (CSUR)* **2021**, *54*, 1–34.
216. Yang, L.; Shami, A. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing* **2020**, *415*, 295–316.
217. Balasubramaniam, D.; Jefferson, C.; Kotthoff, L.; Miguel, I.; Nightingale, P. An automated approach to generating efficient constraint solvers. 2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012, pp. 661–671.
218. Abdollahi, M.; Mashhadi, S.; Sabzalizadeh, R.; Mirzaei, A.; Elahi, M.; Baharloo, M.; Baniasadi, A. IODnet: Indoor/Outdoor Telecommunication Signal Detection through Deep Neural Network. 2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC). IEEE, 2023, pp. 134–141.
219. Mashhadi, S.; Diyanat, A.; Abdollahi, M.; Baniasadi, A. DSP: A Deep Neural Network Approach for Serving Cell Positioning in Mobile Networks. 2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM). IEEE, 2023, pp. 1–6.
220. Abdollahi, M.; Sabzalizadeh, R.; Javadinia, S.; Mashhadi, S.; Mehrizi, S.S.; Baniasadi, A. Automatic Modulation Classification for NLOS 5G Signals with Deep Learning Approaches. 2023 10th International Conference on Wireless Networks and Mobile Communications (WINCOM). IEEE, 2023, pp. 1–6.
221. Yoo, H.J.; Lee, K.; Kim, J.K. *Low-power noc for high-performance soc design*; CRC press, 2018.
222. Baharloo, M.; Aligholipour, R.; Abdollahi, M.; Khonsari, A. ChangeSUB: a power efficient multiple network-on-chip architecture. *Computers & Electrical Engineering* **2020**, *83*, 106578.
223. Yenugula, M. Data Center Power Management Using Neural Network. *International Journal of Advanced Academic Studies*, *3*, 320–25.
224. Kose, N.A.; Jinad, R.; Rasheed, A.; Shashidhar, N.; Baza, M.; Alshahrani, H. Detection of Malicious Threats Exploiting Clock-Gating Hardware Using Machine Learning. *Sensors* **2024**, *24*, 983.
225. Wang, Y.; Sheng, M.; Wang, X.; Wang, L.; Li, J. Mobile-edge computing: Partial computation offloading using dynamic voltage scaling. *IEEE Transactions on Communications* **2016**, *64*, 4268–4282.
226. Joshi, S.; Li, D.; Ogrenci-Memik, S.; Deptuch, G.; Hoff, J.; Jindariani, S.; Liu, T.; Olsen, J.; Tran, N. Multi-Vdd design for content addressable memories (CAM): A power-delay optimization analysis. *Journal of Low Power Electronics and Applications* **2018**, *8*, 25.
227. Tiwari, A.; Bisht, M.R. Leakage Power Reduction in CMOS VLSI Circuits using Advance Leakage Reduction Method. *International Journal for Research in Applied Science and Engineering Technology* **2021**, *9*, 962–966.
228. Pathak, A.; Sachan, D.; Peta, H.; Goswami, M. A modified SRAM based low power memory design. 2016 29th international conference on VLSI design and 2016 15th international conference on embedded systems (VLSID). IEEE, 2016, pp. 122–127.

 doi:10.20944/preprints202411.0156.v1

229. Birla, S.; Singh, N.; Shukla, N. Low-power memory design for IoT-enabled systems: Part 2. In *Electrical and Electronic Devices, Circuits and Materials*; CRC Press, 2021; pp. 63–80.
230. Cao, R.; Yang, Y.; Gu, H.; Huang, L. A thermal-aware power allocation method for optical network-on-chip. *IEEE Access* **2018**, *6*, 61176–61183.
231. Dehghani, F.; Mohammadi, S.; Barekatain, B.; Abdollahi, M. Power loss analysis in thermally-tuned nanophotonic switch for on-chip interconnect. *Nano Communication Networks* **2020**, *26*, 100323.
232. Bhasker, J.; Chadha, R. *Static timing analysis for nanometer designs: A practical approach*; Springer Science & Business Media, 2009.
233. Willis, R. Critical path analysis and resource constrained project scheduling—theory and practice. *European Journal of Operational Research* **1985**, *21*, 149–155.
234. Kao, C.C. Clock skew minimization in multiple dynamic supply voltage with adjustable delay buffers restriction. *Journal of Signal Processing Systems* **2015**, *79*, 99–104.
235. Hatture, S.; Dhage, S. Multi-clock domain synchronizers. 2015 International Conference on Computation of Power, Energy, Information and Communication (ICCPEIC). IEEE, 2015, pp. 0403–0408.
236. Saboori, E.; Abdi, S. Rapid design space exploration of multi-clock domain MPSoCs with Hybrid Prototyping. 2016 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE). IEEE, 2016, pp. 1–6.
237. Chentouf, M.; Ismaili, Z.E.A.A. A PUS based nets weighting mechanism for power, hold, and setup timing optimization. *Integration* **2022**, *84*, 122–130.
238. Wang, C.Y.; Liao, H.Y.M.; Yeh, I.H. Designing network design strategies through gradient path analysis. *arXiv preprint arXiv:2211.04800* **2022**.
239. Mirhoseini, A.; Goldie, A.; Yazgan, M.; Jiang, J.W.; Songhori, E.; Wang, S.; Lee, Y.J.; Johnson, E.; Pathak, O.; Nazi, A.; others. A graph placement methodology for fast chip design. *Nature* **2021**, *594*, 207–212.
240. Dey, S.; Nandi, S.; Trivedi, G. PowerPlanningDL: Reliability-aware framework for on-chip power grid design using deep learning. 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). IEEE, 2020, pp. 1520–1525.
241. Szentimrey, H.; Al-Hyari, A.; Foxcroft, J.; Martin, T.; Noel, D.; Grewal, G.; Areibi, S. Machine learning for congestion management and routability prediction within FPGA placement. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* **2020**, *25*, 1–25.
242. Lin, J.M.; Chang, W.Y.; Hsieh, H.Y.; Shyu, Y.T.; Chang, Y.J.; Lu, J.M. Thermal-aware floorplanning and TSV-planning for mixed-type modules in a fixed-outline 3-D IC. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **2021**, *29*, 1652–1664.
243. Guan, W.; Tang, X.; Lu, H.; Zhang, Y.; Zhang, Y. Thermal-Aware Fixed-Outline 3-D IC Floorplanning: An End-to-End Learning-Based Approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* **2023**.
244. Kim, D.; Kim, M.; Hur, J.; Lee, J.; Cho, J.; Kang, S. TA3D: Timing-Aware 3D IC Partitioning and Placement by Optimizing the Critical Path. Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD, 2024, pp. 1–7.
245. Xu, Q.; Rocha, R.T.; Algoos, Y.; Feron, E.; Younis, M.I. Design, simulation, and testing of a tunable MEMS multi-threshold inertial switch. *Microsystems & Nanoengineering* **2024**, *10*, 31.
246. Hosseini, S.A.; Roosta, E. A novel technique to produce logic '1'in multi-threshold ternary circuits design. *Circuits, Systems, and Signal Processing* **2021**, *40*, 1152–1165.
247. Haj-Yahya, J.; Alser, M.; Kim, J.; Yağlıkçı, A.G.; Vijaykumar, N.; Rotem, E.; Mutlu, O. SysScale: Exploiting multi-domain dynamic voltage and frequency scaling for energy efficient mobile processors. 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). IEEE, 2020, pp. 227–240.
248. Tsou, W.J.; Yang, W.H.; Lin, J.H.; Chen, H.; Chen, K.H.; Wey, C.L.; Lin, Y.H.; Lin, S.R.; Tsai, T.Y. 20.2 digital low-dropout regulator with anti PVT-variation technique for dynamic voltage scaling and adaptive voltage scaling multicore processor. 2017 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2017, pp. 338–339.
249. Lungu, A.; Bose, P.; Buyuktosunoglu, A.; Sorin, D.J. Dynamic power gating with quality guarantees. Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design, 2009, pp. 377–382.
250. Jahanirad, H. Dynamic power-gating for leakage power reduction in FPGAs. *Frontiers of Information Technology & Electronic Engineering* **2023**, *24*, 582–598.

 doi:10.20944/preprints202411.0156.v1

251. Scarabottolo, I.; Ansaloni, G.; Constantinides, G.A.; Pozzi, L.; Reda, S. Approximate logic synthesis: A survey. *Proceedings of the IEEE* **2020**, *108*, 2195–2213.
252. Wu, J.; Zhang, Y.; Zukerman, M.; Yung, E.K.N. Energy-efficient base-stations sleep-mode techniques in green cellular networks: A survey. *IEEE communications surveys & tutorials* **2015**, *17*, 803–826.
253. Ning, S.; Zhu, H.; Feng, C.; Gu, J.; Jiang, Z.; Ying, Z.; Midkiff, J.; Jain, S.; Hlaing, M.H.; Pan, D.Z.; others. Photonic-Electronic Integrated Circuits for High-Performance Computing and AI Accelerators. *Journal of Lightwave Technology* **2024**.
254. Park, H.; Kim, S. Hardware accelerator systems for artificial intelligence and machine learning. In *Advances in Computers*; Elsevier, 2021; Vol. 122, pp. 51–95.
255. Hu, X.; Li, X.; Huang, H.; Zheng, X.; Xiong, X. Tinna: A tiny accelerator for neural networks with efficient dsp optimization. *IEEE Transactions on Circuits and Systems II: Express Briefs* **2022**, *69*, 2301–2305.
256. Liu, S.; Cao, Y.; Sun, S. Mapping and optimization method of SpMV on Multi-DSP accelerator. *Electronics* **2022**, *11*, 3699.
257. Dai, K.; Xie, Z.; Liu, S. DCP-CNN: Efficient Acceleration of CNNs With Dynamic Computing Parallelism on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2024**.
258. Zacharopoulos, G.; Ejjeh, A.; Jing, Y.; Yang, E.Y.; Jia, T.; Brumar, I.; Intan, J.; Huzaifa, M.; Adve, S.; Adve, V.; others. Trireme: Exploration of hierarchical multi-level parallelism for hardware acceleration. *ACM Transactions on Embedded Computing Systems* **2023**, *22*, 1–23.
259. Jamilan, S.; Abdollahi, M.; Mohammadi, S. Cache energy management through dynamic reconfiguration approach in opto-electrical noc. 2017 25th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP). IEEE, 2017, pp. 576–583.
260. Sanca, V.; Ailamaki, A. Post-Moore's Law Fusion: High-Bandwidth Memory, Accelerators, and Native Half-Precision Processing for CPU-Local Analytics. Joint Workshops at 49th International Conference on Very Large Data Bases (VLDBW'23), 2023.
261. Hur, S.; Na, S.; Kwon, D.; Kim, J.; Boutros, A.; Nurvitadhi, E.; Kim, J. A fast and flexible FPGA-based accelerator for natural language processing neural networks. *ACM Transactions on Architecture and Code Optimization* **2023**, *20*, 1–24.
262. Kabir, E.; Kabir, M.A.; Downey, A.R.; Bakos, J.D.; Andrews, D.; Huang, M. FAMOUS: Flexible Accelerator for the Attention Mechanism of Transformer on UltraScale+ FPGAs. *arXiv preprint arXiv:2409.14023* **2024**.
263. Lee, H.; Lee, J.; Kang, S. A Robust Test Architecture for Low-Power AI Accelerators. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2024**.
264. Lee, S.; Park, J.; Park, S.; Kim, H.; Kang, S. A New Zero-Overhead Test Method for Low-Power AI Accelerators. *IEEE Transactions on Circuits and Systems II: Express Briefs* **2023**.
265. Shah, N.; Meert, W.; Verhelst, M. *Efficient Execution of Irregular Dataflow Graphs: Hardware/Software Co-optimization for Probabilistic AI and Sparse Linear Algebra*; Springer Nature, 2023.
266. Rashidi, B.; Gao, C.; Lu, S.; Wang, Z.; Zhou, C.; Niu, D.; Sun, F. UNICO: Unified Hardware Software Co-Optimization for Robust Neural Network Acceleration. Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, 2023, pp. 77–90.
267. Arman, G. New Approach of IO Cell Placement Addressing Minimized Data and Clock Skews in Top Level. 2023 IEEE East-West Design & Test Symposium (EWDTS). IEEE, 2023, pp. 1–5.
268. Deng, C.; Cai, Y.C.; Zhou, Q. Register clustering methodology for low power clock tree synthesis. *Journal of Computer Science and Technology* **2015**, *30*, 391–403.
269. Kyriakakis, E.; Tange, K.; Reusch, N.; Zaballa, E.O.; Fafoutis, X.; Schoeberl, M.; Dragoni, N. Fault-tolerant clock synchronization using precise time protocol multi-domain aggregation. 2021 IEEE 24th International Symposium on Real-Time Distributed Computing (ISORC). IEEE, 2021, pp. 114–122.
270. Han, K.; Kahng, A.B.; Li, J. Optimal generalized H-tree topology and buffering for high-performance and low-power clock distribution. *Ieee transactions on computer-aided design of integrated circuits and systems* **2018**, *39*, 478–491.
271. Rahman, M.S.; Guo, R.; Kamali, H.M.; Rahman, F.; Farahmandi, F.; Abdel-Moneum, M.; Tehranipoor, M. O'clock: lock the clock via clock-gating for soc ip protection. Proceedings of the 59th ACM/IEEE Design Automation Conference, 2022, pp. 775–780.
272. Hu, K.; Hou, X.; Lin, Z. Advancements In Low-Power Technologies: Clock-Gated Circuits and Beyond. *Highlights in Science, Engineering and Technology* **2024**, *81*, 218–225.

 doi:10.20944/preprints202411.0156.v1

273. Erra, R.; Stine, J.E. Power Reduction of Montgomery Multiplication Architectures Using Clock Gating. 2024 IEEE 67th International Midwest Symposium on Circuits and Systems (MWSCAS). IEEE, 2024, pp. 474–478.
274. Namazi, A.; Safari, S.; Mohammadi, S.; Abdollahi, M. SORT: Semi online reliable task mapping for embedded multi-core systems. *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)* **2019**, *4*, 1–25.
275. Namazi, A.; Abdollahi, M.; Safari, S.; Mohammadi, S.; Daneshtalab, M. Lrtm: Life-time and reliability-aware task mapping approach for heterogeneous multi-core systems. 2018 11th International Workshop on Network on Chip Architectures (NoCArc). IEEE, 2018, pp. 1–6.
276. Abumwais, A.; Obaid, M. Shared Cache Based on Content Addressable Memory in a Multi-Core Architecture. *Computers, Materials & Continua* **2023**, *74*.
277. Bahn, H.; Cho, K. Implications of NVM based storage on memory subsystem management. *Applied Sciences* **2020**, *10*, 999.
278. Sarkar, R.; Abi-Karam, S.; He, Y.; Sathidevi, L.; Hao, C. FlowGNN: A dataflow architecture for real-time workload-agnostic graph neural network inference. 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). IEEE, 2023, pp. 1099–1112.
279. Kenter, T.; Shambhu, A.; Faghih-Naini, S.; Aizinger, V. Algorithm-hardware co-design of a discontinuous Galerkin shallow-water model for a dataflow architecture on FPGA. Proceedings of the Platform for Advanced Scientific Computing Conference, 2021, pp. 1–11.
280. Besta, M.; Kanakagiri, R.; Kwasniewski, G.; Ausavarungnirun, R.; Beránek, J.; Kanellopoulos, K.; Janda, K.; Vonarburg-Shmaria, Z.; Gianinazzi, L.; Stefan, I.; others. Sisa: Set-centric instruction set architecture for graph mining on processing-in-memory systems. MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture, 2021, pp. 282–297.
281. Sahabandu, D.; Mertoguno, J.S.; Poovendran, R. A natural language processing approach for instruction set architecture identification. *IEEE Transactions on Information Forensics and Security* **2023**, *18*, 4086–4099.
282. Baharloo, M.; Abdollahi, M.; Baniasadi, A. System-level reliability assessment of optical network on chip. *Microprocessors and Microsystems* **2023**, *99*, 104843.
283. Abdollahi, M.; Baharloo, M.; Shokouhinia, F.; Ebrahimi, M. RAP-NOC: reliability assessment of photonic network-on-chips, a simulator. Proceedings of the Eight Annual ACM International Conference on Nanoscale Computing and Communication, 2021, pp. 1–7.
284. Hasanzadeh, M.; Abdollahi, M.; Baniasadi, A.; Patooghy, A. Thermo-Attack Resiliency: Addressing a New Vulnerability in Opto-Electrical Network-on-Chips. 2024 25th International Symposium on Quality Electronic Design (ISQED). IEEE, 2024, pp. 1–9.
285. Anuradha, P.; Majumder, P.; Sivaraman, K.; Vignesh, N.A.; Jayakar, A.; Anthonirj, S.; Mallik, S.; Al-Rasheed, A.; Abbas, M.; Soufiene, B.O. Enhancing High-Speed Data Communications: Optimization of Route Controlling Network on Chip Implementation. *IEEE Access* **2024**.
286. Nisa, U.U.; Bashir, J. Towards efficient on-chip communication: A survey on silicon nanophotonics and optical networks-on-chip. *Journal of Systems Architecture* **2024**, p. 103171.
287. Abdollahi, M.; Firouzabadi, Y.; Dehghani, F.; Mohammadi, S. THAMON: Thermal-aware High-performance Application Mapping onto Opto-electrical network-on-chip. *Journal of Systems Architecture* **2021**, *121*, 102315.
288. Bai, C.; Huang, J.; Wei, X.; Ma, Y.; Li, S.; Zheng, H.; Yu, B.; Xie, Y. ArchExplorer: Microarchitecture exploration via bottleneck analysis. Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture, 2023, pp. 268–282.
289. Dave, S.; Nowatzki, T.; Shrivastava, A. Explainable-DSE: An Agile and Explainable Exploration of Efficient HW/SW Codesigns of Deep Learning Accelerators Using Bottleneck Analysis. Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 4, 2023, pp. 87–107.
290. Bernstein, L.; Sludds, A.; Hamerly, R.; Sze, V.; Emer, J.; Englund, D. Freely scalable and reconfigurable optical hardware for deep learning. *Scientific reports* **2021**, *11*, 3144.
291. Jia, H.; Ozatay, M.; Tang, Y.; Valavi, H.; Pathak, R.; Lee, J.; Verma, N. 15.1 a programmable neural-network inference accelerator based on scalable in-memory computing. 2021 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2021, Vol. 64, pp. 236–238.

 doi:10.20944/preprints202411.0156.v1

292. Lakshmanna, K.; Shaik, F.; Gunjan, V.K.; Singh, N.; Kumar, G.; Shafi, R.M. Perimeter degree technique for the reduction of routing congestion during placement in physical design of VLSI circuits. *Complexity* **2022**, *2022*, 8658770.
293. Chen, X.; Liu, G.; Xiong, N.; Su, Y.; Chen, G. A survey of swarm intelligence techniques in VLSI routing problems. *IEEE Access* **2020**, *8*, 26266–26292.
294. Karimullah, S.; Vishnuvardhan, D. Experimental analysis of optimization techniques for placement and routing in Asic design. ICDSMLA 2019: Proceedings of the 1st International Conference on Data Science, Machine Learning and Applications. Springer, 2020, pp. 908–917.
295. Ramesh, S.; Manna, K.; Gogineni, V.C.; Chattopadhyay, S.; Mahapatra, S. Congestion-Aware Vertical Link Placement and Application Mapping Onto Three-Dimensional Network-On-Chip Architectures. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2024**.
296. Rocher-Gonzalez, J.; Escudero-Sahuquillo, J.; Garcia, P.J.; Quiles, F.J. Congestion management in high-performance interconnection networks using adaptive routing notifications. *The Journal of Supercomputing* **2023**, *79*, 7804–7834.
297. Cho, Y.; Kim, H.; Lee, K.; Jo, H.; Lee, H.; Kim, M.; Im, Y. Fast and Real-Time Thermal-Aware Floorplan Methodology for SoC. *IEEE Transactions on Components, Packaging and Manufacturing Technology* **2024**.
298. Cho, Y.; Kim, H.; Lee, K.; Im, Y.; Lee, H.; Kim, M. Thermal Aware Floorplan Optimization of SoC in Mobile Phone. 2023 22nd IEEE Intersociety Conference on Thermal and Thermomechanical Phenomena in Electronic Systems (ITherm). IEEE, 2023, pp. 1–7.
299. Dehghani, F.; Mohammadi, S.; Barekatain, B.; Abdollahi, M. ICES: an innovative crosstalk-efficient 2$\times$ 2 photonic-crystal switch. *Optical and Quantum Electronics* **2021**, *53*, 1–15.
300. Kaur, M.; Singh, G.; Kumar, Y.; others. RF and Crosstalk Characterization of Chip Interconnects Using Finite Element Method. *Indian Journal of Engineering and Materials Sciences (IJEMS)* **2023**, *30*, 132–137.
301. Kashif, M.; Cicek, I. Field-programmable gate array (FPGA) hardware design and implementation ofa new area efficient elliptic curve crypto-processor. *Turkish Journal of Electrical Engineering and Computer Sciences* **2021**, *29*, 2127–2139.
302. Bardon, M.G.; Sherazi, Y.; Jang, D.; Yakimets, D.; Schuddinck, P.; Baert, R.; Mertens, H.; Mattii, L.; Parvais, B.; Mocuta, A.; others. Power-performance trade-offs for lateral nanosheets on ultra-scaled standard cells. 2018 IEEE Symposium on VLSI Technology. IEEE, 2018, pp. 143–144.
303. Gao, X.; Qiao, Q.; Wang, M.; Niu, M.; Liu, H.; Maezawa, M.; Ren, J.; Wang, Z. Design and verification of SFQ cell library for superconducting LSI digital circuits. *IEEE Transactions on Applied Superconductivity* **2021**, *31*, 1–5.
304. Dannan, B.; Grumman, N.; Kuszewski, J.; Vincent, R.; Wu, S.; McCaffrey, W.; Park, A. Improved methodology to accurately perform system level power integrity analysis including an ASIC die. In *presented at DesignCon*; Signal Integrity Journal, 2022; pp. 5–7.
305. Meixner, A.; Gullo, L.J. Design for Test and Testability. *Design for Maintainability* **2021**, pp. 245–264.
306. Huhn, S.; Drechsler, R. *Design for Testability, Debug and Reliability*; Springer, 2021.
307. Deshpande, N.; Sowmya, K. A review on ASIC synthesis flow employing two industry standard tools. *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) ICEECT–2020* **2020**, *8*.
308. Taraate, V. ASIC Design and Synthesis. *Springer Nature* **2021**.
309. Golshan, K. *The Art of Timing Closure*; Springer, 2020.
310. Sariki, A.; Sai, G.M.V.; Khosla, M.; Raj, B. ASIC Design using Post Route ECO Methodologies for Timing Closure and Power Optimization. *International Journal of Microsystems and IoT* **2023**.
311. Lau, J.H. Recent advances and trends in advanced packaging. *IEEE Transactions on Components, Packaging and Manufacturing Technology* **2022**, *12*, 228–252.
312. Abdollahi, M.; Mohammadi, S. Vulnerability assessment of fault-tolerant optical network-on-chips. *Journal of Parallel and Distributed Computing* **2020**, *145*, 140–159.
313. Hiller, M.; Kürzinger, L.; Sigl, G. Review of error correction for PUFs and evaluation on state-of-the-art FPGAs. *Journal of cryptographic engineering* **2020**, *10*, 229–247.
314. Djambazova, E.; Andreev, R. Redundancy Management in Dependable Distributed Real-Time Systems. In *j. Problems of Engineering Cybernetics and Robotics*; Prof. Marin Drinov Publishing House of Bulgarian Academy of Sciences, 2023; Vol. 79, pp. 37–54.

315. Oszczypała, M.; Ziółkowski, J.; Małachowski, J. Redundancy allocation problem in repairable k-out-of-n systems with cold, warm, and hot standby: A genetic algorithm for availability optimization. *Applied Soft Computing* **2024**, *165*, 112041.
316. Hantos, G.; Flynn, D.; Desmulliez, M.P. Built-in self-test (BIST) methods for MEMS: A review. *Micromachines* **2020**, *12*, 40.
317. Li, M.; Lin, Y.; Gupta, S. Built in self test (BIST) for RSFQ circuits. 2024 IEEE 42nd VLSI Test Symposium (VTS). IEEE, 2024, pp. 1–7.
318. Verducci, O.; Oliveira, D.L.; Batista, G. Fault-tolerant finite state machine quasi delay insensitive in commercial FPGA devices. 2022 IEEE 13th Latin America Symposium on Circuits and System (LASCAS). IEEE, 2022, pp. 1–4.
319. Salauyou, V. Fault Detection of Moore Finite State Machines by Structural Models. International Conference on Computer Information Systems and Industrial Management. Springer, 2023, pp. 394–409.
320. Pavan Kumar, M.; Lorenzo, R. A review on radiation-hardened memory cells for space and terrestrial applications. *International journal of circuit theory and applications* **2023**, *51*, 475–499.
321. Lee, M.; Cho, S.; Lee, N.; Kim, J. New radiation-hardened design of a cmos instrumentation amplifier and its tolerant characteristic analysis. *Electronics* **2020**, *9*, 388.
322. Wang, Z.; Chen, L.; Wang, S.; Zhou, J.; Tian, C.; Feng, H. AIP-SEM: An Efficient ML-Boost In-Place Soft Error Mitigation Method for SRAM-Based FPGA. 2024 2nd International Symposium of Electronics Design Automation (ISEDA). IEEE, 2024, pp. 351–354.
323. Xie, Y.; Qiao, T.; Xie, Y.; Chen, H. Soft error mitigation and recovery of SRAM-based FPGAs using brain-inspired hybrid-grained scrubbing mechanism. *Frontiers in Computational Neuroscience* **2023**, *17*, 1268374.
324. Xu, F.; Ding, N.; Li, N.; Liu, L.; Hou, N.; Xu, N.; Guo, W.; Tian, L.; Xu, H.; Wu, C.M.L.; others. A review of bearing failure Modes, mechanisms and causes. *Engineering Failure Analysis* **2023**, p. 107518.
325. Huang, J.; You, J.X.; Liu, H.C.; Song, M.S. Failure mode and effect analysis improvement: A systematic literature review and future research agenda. *Reliability Engineering & System Safety* **2020**, *199*, 106885.
326. Chen, B.; Zhang, F.; Nguyen, A.; Zan, D.; Lin, Z.; Lou, J.G.; Chen, W. Codet: Code generation with generated tests. *arXiv preprint arXiv:2207.10397* **2022**.
327. Unno, H.; Terauchi, T.; Koskinen, E. Constraint-based relational verification. International Conference on Computer Aided Verification. Springer, 2021, pp. 742–766.
328. Jha, C.K.; Qayyum, K.; Coşkun, K.Ç.; Singh, S.; Hassan, M.; Leupers, R.; Merchant, F.; Drechsler, R. veriSIMPLER: An Automated Formal Verification Methodology for SIMPLER MAGIC Design Style Based In-Memory Computing. *IEEE Transactions on Circuits and Systems I: Regular Papers* **2024**.
329. Coudert, S.; Apvrille, L.; Sultan, B.; Hotescu, O.; de Saqui-Sannes, P. Incremental and Formal Verification of SysML Models. *SN Computer Science* **2024**, *5*, 714.
330. Ayalasomayajula, A.; Farzana, N.; Tehranipoor, M.; Farahmandi, F. Automatic Asset Identification for Assertion-Based SoC Security Verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **2024**.
331. Rostami, H.; Hosseini, M.; Azarpeyvand, A.; Iman, M.R.H.; Ghasempouri, T. Automatic High Functional Coverage Stimuli Generation for Assertion-based Verification. 2024 IEEE 30th International Symposium on On-Line Testing and Robust System Design (IOLTS). IEEE, 2024, pp. 1–7.
332. Tian, K.; Mitchell, E.; Yao, H.; Manning, C.D.; Finn, C. Fine-tuning language models for factuality. *arXiv preprint arXiv:2311.08401* **2023**.
333. Yang, Z. Scalable Equivalence Checking for Behavioral Synthesis. PhD thesis, Computer Science Department, Portland State University, 2015.
334. Aboudeif, R.A.H. Design and Implementation of UVM-based Verification Framework for Deep Learning Accelerators. Master's thesis, School of Sciences and Engineering, The American University in Cairo, 2024.