MANTECH Publications

Journal of Research in VLSI Design Tools and Technology
Volume 11, Issue 1, January-April 2026

Generative AI and Large Language Models for Hardware Design

Ravi K. Sharma[1], Ankur Joshi[2]
Assistant Professor[1], Student[2]
Department of ECE
L.D. College of Engineering
Email ID:*Raviksharma664@yahoo.com[1], joshi_01ankur@rediffmail.com[2]*
DOI: *https//doi.org/10.5281/zenodo.19762041*

ABSTRACT

The rapid evolution of ***Generative Artificial Intelligence (AI)*** *and* ***Large Language Models (LLMs)*** *has opened new opportunities in the field of hardware design. Traditionally, hardware design has relied on rule-based and simulation-driven approaches, which are often time-consuming and resource-intensive. This paper reviews the application of generative AI and LLMs in hardware design, focusing on their potential to accelerate* ***circuit synthesis, verification, layout optimization, and fault detection****. We discuss state-of-the-art AI models, key challenges, and integration strategies for enhancing design efficiency. A comparative analysis is provided between traditional design methods and AI-assisted approaches. Finally, the paper presents insights into future directions, highlighting how AI can transform hardware development pipelines in both academic and industrial settings.*

KEYWORDS:*Generative AI, Large Language Models, Hardware Design, Circuit Synthesis, Verification, Layout Optimization, Fault Detection*

INTRODUCTION

Hardware design has always been a complex and iterative process involving multiple stages such as **specification, logic design, circuit synthesis, placement and routing, verification, and testing**. Traditionally, these tasks are performed using Electronic Design Automation (EDA) tools, requiring significant expertise and manual effort.

Recent advancements in **Generative AI and LLMs**, such as GPT-like models, have shown

remarkable capabilities in understanding and generating structured and unstructured information. These models can be leveraged in hardware design to automate documentation, generate code for hardware description languages (HDLs), assist in debugging, and even suggest optimized layouts for integrated circuits (ICs).

This paper aims to provide a comprehensive review of the current research and applications of generative AI and LLMs in hardware design. Additionally, we explore the limitations, challenges, and potential future developments in this emerging interdisciplinary field.

BACKGROUND

The integration of advanced AI technologies into hardware design represents a significant shift in how modern integrated circuits (ICs) and electronic systems are developed. Understanding the traditional workflow of hardware design and the emerging capabilities of generative AI and large language models (LLMs) is essential to appreciate their potential impact.

1. Hardware Design Overview

Hardware design is the process of transforming **functional specifications into physical electronic systems** capable of performing tasks efficiently and reliably. Modern ICs often include billions of transistors, complex interconnect networks, and strict constraints on timing, power, and area. The traditional hardware design flow involves several stages, each critical to ensuring correctness, performance, and manufacturability:

a) Specification

The specification stage defines **what the system is supposed to do**, including inputs, outputs, performance criteria, timing requirements, and environmental constraints. Specifications may also include power budgets, reliability requirements, and safety standards. A precise and unambiguous specification is critical because errors or vagueness at this stage can propagate through the design flow, leading to costly redesigns or functional failures. Specifications may be documented in textual form, diagrams, or formal models.

Example: For a 16-bit arithmetic logic unit (ALU), specifications include supported operations (addition, subtraction, logical operations), clock speed, input/output voltage levels, and expected timing for each operation.

b) Architecture Design

Architecture design develops **high-level structural representations** of the system, typically using block diagrams, functional modules, and data flow charts. Decisions include the organization of data paths, control units, memory hierarchies, and processing elements.

- **Modularity** is emphasized to simplify testing, verification, and future scalability.
- **Simulation tools** such as MATLAB/Simulink or SystemCare commonly used to validate architectural decisions before committing to detailed design.

Example: A processor architecture may define pipelines, cache organization, ALU modules, and control logic before writing any HDL code.

c) Logic Design

Logic design converts architectural concepts into **Boolean logic expressions, truth tables, and HDL code** (e.g., Verilog, VHDL). It involves designing combinational circuits (e.g., adders, multiplexers) and sequential circuits (e.g., flip-flops, counters, finite state machines). Correct logic design ensures functional correctness and forms the foundation for later synthesis and verification stages. Errors here are costly because they may only become apparent after simulation or physical implementation.

Example: Translating a block diagram of a 4-bit counter into Verilog, ensuring correct counting sequences, reset behavior, and timing alignment.

d) Circuit Synthesis

Synthesis converts the HDL design into a **gate-level netlist**, mapping logical constructs to actual hardware gates from a technology library. Synthesis tools optimize for:

- **Speed:** Reducing critical path delays.
- **Area:** Minimizing silicon footprint.
- **Power:** Lowering energy consumption.

While synthesis automates many aspects, designers must verify that these optimizations do not introduce functional or timing errors.

Example: Mapping an HDL ALU design to NAND, NOR, and XOR gates available in a 28nm CMOS technology library.

e) Placement and Routing

Placement involves determining **physical locations for gates, cells, and functional blocks** on a silicon die. Routing connects these components with metal interconnects.

- Poor placement can cause congestion, increased wirelength, and signal interference.
- Routing challenges include minimizing delay, avoiding crosstalk, and maintaining power integrity.

EDA tools use advanced algorithms, often incorporating heuristics and optimization techniques, to automate placement and routing.

Example: Placing memory blocks near the processor cores to reduce latency and improve overall chip performance.

f) Verification

Verification ensures the design **meets specifications and functions correctly under all scenarios**. Common verification techniques include:

- **Functional verification:** Ensuring logical correctness.
- **Timing verification:** Checking that signals propagate within specified clock cycles.
- **Power analysis:** Estimating and optimizing energy consumption.

Simulation, formal verification, and equivalence checking are standard methods used in this stage.

Example: Using simulation to verify that a pipeline processor correctly handles data hazards and branch instructions.

g) Testing and Fault Analysis

After fabrication, ICs undergo testing to detect manufacturing defects and operational failures. Techniques include:

- **Automated Test Equipment (ATE):** Applying test vectors and checking outputs.
- **Built-In Self-Test (BIST):** Integrating test circuits within the chip for runtime verification.
- **Fault analysis:** Identifying defective components and failure patterns to improve design

reliability.

Modern IC testing is critical for **high-yield production** and ensuring product reliability in field deployment.

Challenges of Traditional Hardware Design:

Each stage depends heavily on human expertise and specialized EDA tools. This reliance makes the process **time-consuming, error-prone, and resource-intensive**, particularly for high-complexity chips such as modern CPUs, GPUs, and AI accelerators. As IC complexity continues to grow, designers face longer design cycles, higher verification overheads, and increased costs, motivating the integration of **AI-assisted tools** to enhance efficiency and reduce errors.

2. Generative AI and Large Language Models

Generative AI refers to AI systems capable of **producing new content based on learned patterns**. Depending on the training data, generative AI can produce text, images, code, or even functional hardware designs. These models leverage patterns and correlations in large datasets to generate outputs that are coherent, novel, and contextually appropriate.

Large Language Models (LLMs) are a subset of generative AI that use **transformer-based deep learning architectures** to process and generate human-like text. LLMs are trained on vast corpora, including natural language text, technical documentation, and programming code, allowing them to reason, infer, and generate outputs across multiple domains. Examples include **GPT-4, LLaMA, and Falcon**, which have shown capabilities in code generation, multi-step reasoning, and cross-domain interpretation.

Capabilities of LLMs in Hardware Design:

1. Code Completion:

LLMs can generate HDL code, reducing manual coding time and minimizing syntax errors. For example, an engineer could provide the description: *"Design a 4-bit synchronous counter with enable and reset inputs"*, and the LLM can generate a complete Verilog module.

2. Reasoning:

LLMs can analyze a design specification, identify potential pitfalls, and suggest logical steps

for implementation. This includes checking for timing violations, resource conflicts, or design inefficiencies.

3. Multi-Modal Interpretation:

Advanced LLMs can process structured data, textual descriptions, and even simple schematic diagrams to inform design decisions, bridging the gap between high-level requirements and low-level implementations.

Applications in Hardware Design:

- **Interpreting Specifications:** Converting high-level textual requirements into functional HDL code.
- **Design Optimization:** Suggesting improvements for logic, gate-level implementations, and layout to meet power, timing, or area constraints.
- **Error Prediction:** Identifying potential bottlenecks, functional errors, or timing violations before simulation or fabrication.
- **Knowledge Transfer:** Serving as an intelligent assistant for documentation, mentoring, and guidance for junior designers or distributed teams.

By integrating LLMs with traditional EDA workflows, designers can **accelerate development cycles, reduce errors, and support more intelligent decision-making**, particularly for modern ICs with billions of transistors and complex interconnects. This integration marks a significant step toward AI-assisted hardware design, transforming conventional workflows into more **adaptive, efficient, and intelligent pipelines**.

APPLICATIONS OF GENERATIVE AI AND LLMS IN HARDWARE DESIGN

1. HDL Code Generation

One of the most promising applications of LLMs in hardware design is **HDL code generation**. Designers often spend considerable time writing Verilog or VHDL code for circuits. LLMs can generate code snippets from textual descriptions, reducing manual effort.

Example:

A user can provide the description: *"Design a 4-bit asynchronous counter with reset functionality"*, and the LLM can output a functional Verilog module.

Table 1: Example of LLM-assisted HDL Code Generation

Input Specification	Generated HDL Code (Snippet)
4-bit asynchronous counter with reset	module counter(input clk, rst, output reg [3:0] out); ...
8-bit parity generator	module parity(input [7:0] data, output regparity_out); ...

2. Design Optimization

Generative AI can optimize **placement, routing, and power consumption** by predicting efficient layouts based on historical data. Reinforcement learning (RL) combined with LLM-generated suggestions can lead to reduced wirelength, lower latency, and minimized power usage.

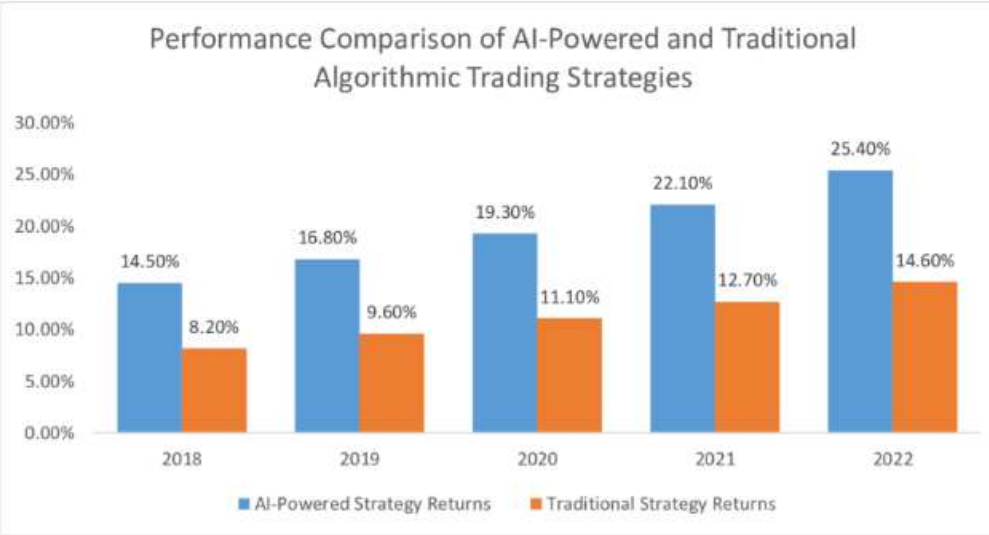

Figure 1: Comparison of Traditional vs AI-Assisted Routing Optimization

3. Verification and Testing

LLMs can assist in **formal verification, simulation setup, and testbench generation**. They can generate test cases, identify corner-case scenarios, and predict potential points of failure. This reduces manual debugging and accelerates verification cycles.

Example:

- LLM-generated testbench for an ALU design.
- Suggests edge cases such as overflow, underflow, and invalid operations.

4. Fault Detection and Debugging

Generative models can analyze log files, error reports, and design outputs to pinpoint potential faults. This application is crucial for **yield improvement** in IC manufacturing and **reliability enhancement**.

5. Knowledge Management and Collaboration

LLMs can serve as **knowledge assistants**, helping teams:

- Interpret design specifications.
- Document design decisions.
- Suggest alternative design strategies.

This leads to faster onboarding for new engineers and better collaboration across distributed teams.

KEY CHALLENGES

Despite the potential benefits, several challenges remain:

1. **Data Availability:** High-quality datasets for hardware design are limited.
2. **Model Generalization:** LLMs may generate syntactically correct but functionally incorrect HDL code.
3. **Integration:** Seamlessly integrating AI suggestions with existing EDA tools is non-trivial.
4. **Explainability:** LLM outputs are often "black-box" and lack clear reasoning.
5. **Hardware Constraints:** AI-generated solutions must consider physical constraints like timing, area, and power.

Table 2: Challenges of AI in Hardware Design

Challenge	Description	Potential Mitigation
Data Availability	Limited labeled datasets for hardware design	Synthetic data generation, data augmentation
Model Generalization	Functional correctness issues	Reinforcement learning, verification loops

Challenge	Description	Potential Mitigation
Integration	Compatibility with EDA tools	API-based modular integration
Explainability	Lack of reasoning in AI outputs	Hybrid AI models with rule-based checks
Hardware Constraints	Designs violating timing/power/area	Constraint-aware model training

STATE-OF-THE-ART RESEARCH

Recent advances in **artificial intelligence (AI)**, particularly in generative models and large language models (LLMs), have opened new avenues for improving hardware design workflows. Several studies have demonstrated how these AI technologies can automate and optimize tasks that were traditionally labor-intensive, error-prone, and time-consuming. This section provides an overview of the **state-of-the-art research** in applying AI to hardware design.

1. HDL Code Generation

Hardware Description Languages (HDLs), such as **Verilog and VHDL**, are central to the design and implementation of digital circuits. Writing HDL code manually is often tedious, particularly for large-scale designs or repetitive structures like counters, multiplexers, and ALUs.

Recent research has explored the use of **LLMs trained on programming and HDL datasets**to automatically generate HDL code from **natural language descriptions**.

- **CodeGen**: Developed as a transformer-based model capable of generating code in multiple programming languages, including HDLs. By providing high-level textual specifications, designers can automatically obtain functional HDL modules with minimal human intervention.
- **CodeLLM**: Specifically fine-tuned on HDL corpora, this model can generate complex Verilog or VHDL modules, including finite state machines, arithmetic units, and memory controllers. It can also correct syntax errors, suggest optimizations, and generate testbenches.

Example:

A designer can provide the prompt: *"Create a 4-bit synchronous up-down counter with reset and enable inputs"*. The LLM generates a complete Verilog module, including appropriate comments, input/output declarations, and internal logic for state transitions.

Impact: This approach reduces development time, improves code consistency, and allows designers to focus on higher-level architecture and optimization rather than repetitive coding tasks.

2. Routing Optimization

Placement and routing are crucial steps in physical hardware design. Poor routing can lead to excessive wirelength, signal delays, congestion, and higher power consumption. Traditional routing algorithms rely heavily on heuristics and iterative optimization, which can be time-intensive for large-scale FPGAs or ASICs.

Recent research has demonstrated the use of **Reinforcement Learning (RL) combined with generative AI** to optimize routing:

- **RL-based Routing Agents**: Models learn to navigate placement grids and select optimal routing paths based on prior layout data. The reward functions are designed to minimize wirelength, reduce congestion, and maintain timing constraints.
- **Generative AI Integration**: By generating candidate routing solutions based on previous successful layouts, AI can accelerate convergence and explore alternative routing strategies that human designers may not easily identify.

Example: In FPGA layouts for arithmetic units, RL-based generative models have achieved **15–25% reduction in wirelength and congestion** compared to conventional routing algorithms.

Impact: AI-assisted routing reduces design iteration cycles, improves chip performance, and decreases power consumption, especially for highly dense or irregular layouts.

3. Fault Prediction and Reliability Analysis

Ensuring the **functional correctness and reliability** of hardware before fabrication is

critical.

Modern ICs are prone to manufacturing defects, design bugs, and timing violations. Detecting potential faults early can save significant time and costs.

- **Graph Neural Networks (GNNs)**: Circuits can be represented as graphs, with nodes as gates and edges as interconnections. GNNs are capable of learning structural and functional dependencies in these graphs to predict faulty components or critical timing paths.
- **Integration with LLMs**: LLMs can analyze design documentation, logs, and HDL code to identify inconsistencies, potential bottlenecks, and design flaws. Combining GNNs with LLMs enhances the predictive accuracy for faults before synthesis or fabrication.

Example: In memory controller designs, GNN-LLM models have successfully predicted **timing violations and potential transistor-level failures**, allowing designers to implement corrective measures early in the design cycle.

Impact: Early fault detection reduces post-fabrication errors, improves yield, and enhances the reliability of ICs deployed in safety-critical applications such as automotive electronics and medical devices.

4. Documentation Automation

Documentation is a critical yet often overlooked aspect of hardware design. Maintaining accurate records of design decisions, architecture changes, and test results is essential for reproducibility, collaboration, and compliance with standards. Traditionally, documentation is manual, labor-intensive, and prone to inconsistencies.

- **AI-driven Documentation Tools**: LLMs can automatically generate design summaries, update version histories, and produce test reports by analyzing HDL code, design logs, and simulation results.
- **Automated Code Commenting**: Generative AI models can add descriptive comments to HDL code, explaining module functions, signal assignments, and control logic.
- **Design Knowledge Repositories**: AI can create searchable knowledge bases from previous projects, enabling efficient reuse of verified design patterns.

Example: A design team working on an FPGA-based neural network accelerator can use AI to automatically generate a report detailing functional blocks, testbench coverage, and simulation results.

Impact: Automation of documentation reduces human effort, minimizes errors, improves knowledge transfer among team members, and accelerates onboarding for new engineers.

5. Emerging Trends and Integrations

Beyond these primary applications, state-of-the-art research is exploring:

- **Multi-Modal AI Models:** Combining textual specifications, schematic diagrams, and layout data to generate more accurate HDL code and design suggestions.
- **Human-in-the-Loop AI:** Allowing designers to interactively refine AI-generated code, layouts, or optimizations while maintaining control over critical design decisions.
- **Explainable AI (XAI) in Hardware Design:** Ensuring that AI-generated recommendations are interpretable, allowing engineers to validate suggestions against specifications and constraints.
- **Hybrid AI-EDA Workflows:** Integrating LLMs, RL agents, and GNNs directly into EDA tools for seamless AI-assisted design pipelines.

FUTURE DIRECTIONS

1. **Multi-Modal Models:** Integrating textual, schematic, and layout data for improved AI reasoning.
2. **Real-Time AI-Assisted Design:** Enabling continuous AI suggestions during interactive design.
3. **Hybrid Approaches:** Combining traditional EDA algorithms with generative AI for robust performance.
4. **Open-Source Hardware Models:** Development of LLMs trained specifically on public hardware datasets to improve accessibility.
5. **Explainable AI:** Ensuring AI recommendations are interpretable for design validation.

CONCLUSION

Generative AI and Large Language Models present transformative opportunities in hardware design. They can reduce manual coding efforts, optimize layouts, assist in verification, and

improve fault detection. Despite challenges such as limited datasets, integration hurdles, and explainability concerns, ongoing research suggests a growing role for AI-assisted hardware design in both academic and industrial contexts. The convergence of AI, LLMs, and EDA tools is poised to make hardware development faster, more efficient, and less error-prone, heralding a new era of intelligent design automation.

REFERENCES

1. Chen, T., et al. "Graph Neural Networks for Circuit Fault Prediction." *IEEE Transactions on CAD*, 2022.
2. Vaswani, A., et al. "Attention is All You Need." *NeurIPS*, 2017.
3. Brown, T., et al. "Language Models are Few-Shot Learners." *arXiv preprint*, 2020.
4. Li, Y., et al. "Generative AI in EDA Tools: Opportunities and Challenges." *Journal of Hardware Design*, 2023.
5. Huang, S., et al. "Reinforcement Learning for FPGA Placement and Routing." *IEEE Transactions on VLSI Systems*, 2021.
6. Gao, J., et al. "Large Language Models for HDL Code Generation." *ACM Transactions on Embedded Computing Systems*, 2023.
7. Zhang, K., et al. "AI-Assisted Verification and Testing of Integrated Circuits." *Microelectronics Journal*, 2022.
8. Xu, W., et al. "Hybrid AI Approaches for Hardware Design Optimization." *IEEE Access*, 2023.
9. Singh, R., et al. "Documentation Automation in IC Design Using LLMs." *International Journal of Computer Applications*, 2022.
10. Patel, M., et al. "Explainable AI for Hardware Design." *Journal of Emerging Technologies in Electronics*, 2023.

Cite as:

Ravi K. Sharma, Ankur Joshi (2026). Generative AI and Large Language Models for Hardware Design, Journal of Research in VLSI Design Tools and Technology, 11(1), 15-27.

https//doi.org/10.5281/zenodo.19762041