MANTECH Publications

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

# *Generative AI Techniques for VLSI Design Automation*

***Mamta Tiwari[1], Alok S Pathak[2], Chandeep Yadav[3]***

*Associate Professor[1], PG Scholars[2, 3]*

*Department of ECE*

*P.K. Institute of Technology*

***Email ID:*** *Mamtatiwari24@gmail.com[1], pathakalock5s5@yahoo.com[2], Yadav_chandeep332@rediffmail.com[3]*

***DOI:*** *https://doi.org/10.5281/zenodo.19762076*

***ABSTRACT***

*The design of Very Large Scale Integration (VLSI) circuits has become increasingly complex, requiring high levels of automation to meet the demands of performance, power, and area. Recent advances in Generative Artificial Intelligence (AI) offer promising techniques for automating various stages of VLSI design, including logic synthesis, placement, routing, and verification. This paper presents a comprehensive review of the current state-of-the-art generative AI approaches applied to VLSI design automation, highlighting their potential benefits, limitations, and future research directions. The review also discusses practical considerations, tool integration, and case studies demonstrating the effectiveness of AI-assisted design processes.*

***KEYWORDS:*** *VLSI, Design Automation, Generative AI, Machine Learning, Placement, Routing, HDL Generation*

## INTRODUCTION

VLSI design has witnessed rapid evolution over the past few decades. Traditionally, VLSI design flows involve multiple stages: **specification, logic synthesis, placement, routing, timing analysis, and verification**. Each stage requires careful optimization to balance performance, area, and power consumption. As circuit complexity grows, conventional Electronic Design Automation (EDA) tools struggle to handle large-scale designs efficiently.

Generative AI techniques, including **Generative Adversarial Networks (GANs), Transformer models, and Reinforcement Learning (RL)**, have emerged as powerful

methods for automating design tasks. By learning patterns from large datasets, these models can generate high-quality design solutions with minimal human intervention. The primary objective of this review is to explore these AI techniques, their applications in VLSI, and potential research opportunities.

## LITERATURE REVIEW

### 1. Traditional VLSI Design Automation

Traditional EDA tools rely heavily on heuristic algorithms for placement and routing. These include **Simulated Annealing (SA)** for placement and *A or Dijkstra algorithms** for routing. While effective for moderate designs, these methods often face scalability challenges and require extensive manual tuning.

### 2. AI in VLSI Design

The integration of AI in VLSI is not new. Early works utilized **Artificial Neural Networks (ANNs)** for power estimation and timing prediction. However, with the advent of **deep learning**, AI can now perform generative tasks, including HDL code generation and layout optimization. Notable studies include:

- **HDL Generation Using Transformers**: Models trained on Verilog/VHDL datasets can convert high-level specifications into hardware description code automatically.
- **RL for Placement and Routing**: Reinforcement learning agents learn placement strategies by maximizing performance metrics such as wirelength minimization and congestion reduction.
- **GANs for Layout Generation**: GANs can generate plausible placement and floorplan solutions by learning from existing chip layouts.

## GENERATIVE AI TECHNIQUES IN VLSI

The complexity of modern VLSI circuits has created a need for advanced automation techniques. Generative AI provides several promising approaches, primarily including **Transformer-based HDL generation**, **Reinforcement Learning (RL) for placement and routing**, and **GANs for floorplanning/layout generation**. These techniques help automate tasks that are traditionally time-consuming, such as writing hardware description code, optimizing placement of cells, or generating efficient floorplans.

### 1. Transformer-Based HDL Generation

Transformers, initially developed for natural language processing tasks, have shown remarkable capability in understanding sequences and generating coherent outputs. In VLSI design, these models can learn the syntax and structure of **Verilog and VHDL** from large datasets of existing HDL modules, allowing them to generate new, functionally correct hardware descriptions based on textual prompts.

For example, given a natural language description like:
"Design a 4-bit ripple carry adder with synchronous reset,"
a trained Transformer model can produce HDL code that includes module definition, input/output declarations, and behavioral description. This reduces the manual coding effort for designers and allows rapid prototyping of hardware modules.

**Workflow of Transformer-Based HDL Generation**

1. **Dataset Preparation:** Collect a large dataset of existing HDL modules and documentation, including code snippets, comments, and module descriptions.
2. **Preprocessing:** Tokenize HDL code and natural language prompts. Include both syntactic and semantic context to improve learning.
3. **Model Training:** Train a Transformer (or a variant like GPT or BERT) on HDL sequences to predict next tokens in code generation.
4. **Code Generation:** Provide a natural language prompt; the model outputs HDL code.
5. **Verification:** Generated code is checked for syntax errors and functional correctness using simulation tools.

**Advantages:**

- **Reduces Human Effort:** Designers no longer need to manually write repetitive or boilerplate HDL code.
- **Rapid Prototyping:** New modules can be generated in seconds rather than hours.
- **Educational Value:** Helps students and new designers understand code patterns and design structures.

**Limitations:**

- **Data Dependency:** Requires large, high-quality HDL datasets for effective training.

- **Potential Bugs:** Models may produce subtle logical or syntactical errors that require simulation-based verification.
- **Scalability:** Extremely large designs or multi-module systems may require careful prompt engineering or model fine-tuning.

**Example Use Case**

A research group generated arithmetic units for a 16-bit ALU using a Transformer model. While the initial generated code was syntactically correct, timing simulations revealed minor propagation delays. Iterative refinement using prompts and corrections allowed the model to produce fully functional modules. This highlights both the potential and limitations of Transformers in HDL generation.

## 2. Reinforcement Learning for Placement

Placement is a critical stage in VLSI design where standard cells and macros are positioned on the chip to optimize performance metrics such as **wirelength, congestion, timing, and power**. Traditionally, heuristic algorithms like **Simulated Annealing** or **Force-Directed Placement** are used, but they may struggle with scalability and adaptability for modern complex circuits. **Reinforcement Learning (RL)** offers an adaptive, learning-based approach. In RL, an agent interacts with the environment (the chip layout) and learns placement strategies through trial-and-error, guided by reward signals representing design objectives.

**RL Placement Workflow**

1. **Environment Setup:** The netlist and design constraints are modeled as the RL environment. The environment computes rewards based on placement quality metrics.
2. **Agent Action:** The RL agent proposes positions for standard cells or clusters.
3. **Reward Calculation:** Metrics such as total wirelength, congestion, timing slack, and power consumption are used to compute the reward.
4. **Policy Update:** The agent updates its policy to maximize cumulative rewards, learning optimal placement strategies over multiple episodes.
5. **Evaluation:** The final placement solution is validated using standard EDA tools.

**Advantages:**

- **Adaptive Optimization:** RL agents can learn non-intuitive placement strategies that

improve over time.

- **Multiple Objectives:** Can balance trade-offs between wirelength, timing, and congestion in a single framework.
- **Automation:** Reduces manual tuning compared to heuristic methods.

**Challenges:**

- **Training Complexity:** RL agents require many iterations and high computational resources to converge.
- **Reward Shaping:** Designing a reward function that balances conflicting objectives can be difficult.
- **Generalization:** Policies trained on one type of design may not transfer easily to different architectures.


***Figure 1: RL-based Placement Workflow***

## 3. GANs for Layout Generation

**Generative Adversarial Networks (GANs)**, introduced by Goodfellow et al. in 2014, are a class of generative models where two neural networks — a **generator** and a **discriminator** — compete in a zero-sum game. In the context of VLSI design, GANs can learn patterns from existing chip layouts and generate new, realistic layouts that adhere to physical design rules.

### a) GAN Architecture for Layout Generation

- **Generator:**
  - Takes random noise or a latent vector as input.

- Outputs a candidate layout (placement or floorplan), which includes positions of macros, standard cells, and routing regions.
- The goal is to produce layouts that are indistinguishable from real ones in terms of design metrics (wirelength, congestion, timing, etc.).

- **Discriminator:**
  - Evaluates layouts from the generator and real designs from the training set.
  - Outputs a probability indicating whether a layout is real (from dataset) or fake (from generator).
  - Provides feedback to the generator to improve its outputs over successive iterations.

**Workflow:**

Over multiple training epochs, the generator improves, learning to produce **realistic and feasible layouts** that satisfy design constraints such as area utilization, timing, and wirelength.

**b) Advantages of GAN-Based Layout Generation**

- **Realistic Layouts:** GANs can capture complex spatial patterns and constraints found in real chip layouts.
- **Novel Solutions:** The generator may propose creative layouts that human designers might not consider.
- **Automation:** Reduces manual intervention in early floorplanning stages, speeding up the design cycle.
- **Scalable to Medium Designs:** Effective for small-to-medium scale chips (thousands to tens of thousands of cells).

**c) Limitations and Challenges**

- **Training Instability:** GANs are known to be difficult to train, often requiring careful tuning of generator and discriminator architectures.
- **Large Datasets Required:** Effective training demands large datasets of prior layouts, which may not always be available.
- **Verification Needed:** Generated layouts must be verified using traditional EDA tools to ensure timing, power, and routing constraints are satisfied.

- **Scalability Issues:** For very large designs (millions of cells), training GANs becomes computationally intensive, and generated layouts may fail to satisfy all physical constraints.

### d) Practical Example

A research team applied GANs to generate floorplans for a small CPU core (~5000 cells). The generator produced layouts with:

- **10–15% better area utilization** than baseline heuristic methods.
- Reduced congestion in critical paths by ~12%.
- Comparable timing performance to manually optimized layouts.

However, the team noted that for more complex SoC designs with multiple macros and IP blocks, the GAN struggled without additional constraints or hybrid approaches (e.g., combining GAN output with classical placement refinement).

***Table 1: Comparison of Generative Techniques in VLSI***

| Technique | Application | Pros | Cons |
|---|---|---|---|
| Transformer | HDL Generation | Fast code generation, easy prototyping | Requires large datasets, may generate errors |
| RL | Placement & Routing | Learns optimal strategies, adaptive | High training cost, complex reward design |
| GAN | Floorplan/Layout | Realistic layouts, novel solutions | Training instability, requires labeled layouts |

## 4. Hybrid Approaches

Recent research suggests combining multiple generative techniques for better results. For example, an RL agent may propose a placement solution that is then refined using a GAN-based layout generator. Such hybrid methods have shown superior performance in minimizing wirelength and congestion while maintaining timing constraints.

## CASE STUDIES

To understand the practical applications of generative AI in VLSI design, several case studies have been conducted at mid-tier research institutions and small-scale industrial labs. These

studies demonstrate how **Transformers, Reinforcement Learning (RL), and GANs** can improve design productivity, optimize layouts, and reduce manual effort.

### 1. HDL Generation Using Transformers

A research team at a mid-tier Indian college focused on **automating HDL code generation** for standard arithmetic modules, such as adders, multipliers, and simple ALUs.

**Study Setup:**

- Dataset: 10,000+ HDL modules collected from open-source repositories.
- Model: Transformer-based sequence-to-sequence model trained for 50 epochs.
- Task: Generate Verilog modules from natural language descriptions like *"4-bit ripple carry adder with synchronous reset"*.

**Results:**

- **Code Development Time:** Reduced by approximately **40%**, as designers no longer had to manually write repetitive modules.
- **Verification:** Minor functional errors appeared in ~8% of generated modules, mostly due to corner cases in timing or reset behavior.
- **Productivity:** Engineers could focus on verification and optimization instead of boilerplate code writing.

**Insights:**

- Transformers excel in **repetitive and structured code generation**, making them ideal for arithmetic and logic modules.
- Iterative feedback loops (human verification → retraining) improve model accuracy over time.

**Practical Note:** For complex modules like pipelined multipliers, the model required additional fine-tuning and modular prompts to produce correct code, showing the importance of task-specific dataset preparation.

### 2. Placement Optimization Using Reinforcement Learning

In a small-scale ASIC design project, an RL agent was applied to optimize the placement of standard cells and macros on a 5000-cell benchmark.

**Study Setup:**

- **Environment:** Simulated ASIC placement environment modeled with wirelength, congestion, and timing slack as reward signals.
- **Agent:** Deep Q-Network (DQN) with exploration-exploitation strategy.
- **Training:** 5000 iterations over multiple layout episodes.

**Results:**

- **Wirelength Improvement:** Reduced total wirelength by **15%** compared to traditional simulated annealing placement.
- **Timing Performance:** Slight improvements (~5%) in critical path delays due to better placement of high-fanout nets.
- **Adaptability:** RL agent automatically adjusted placement strategy when the netlist structure changed.

***Table 4.1: Placement Optimization Results***

| Metric | Simulated Annealing | RL-Based Placement | Improvement |
|---|---|---|---|
| Total Wirelength | 1200 µm | 1020 µm | 15% |
| Max Congestion | 0.85 | 0.72 | 15.3% |
| Critical Path Delay | 3.2 ns | 3.05 ns | 4.7% |

**Insights:**

- RL provides **adaptive learning**, enabling better performance than heuristic-based placement.
- Training overhead is significant, but once trained, the agent can generalize to similar netlists.

**Practical Note:** For larger-scale ASICs (e.g., >50k cells), hybrid methods combining RL with classical heuristics are recommended to reduce computation time.

### 3. GAN-Based Floorplanning

GANs were applied to generate floorplans for a small CPU core (~5000 cells) to explore the feasibility of **automated layout generation**.

**Study Setup:**

- **Dataset:** Realistic floorplans from open-source CPU designs.
- **Model:** Standard GAN architecture with generator and discriminator, trained for 100 epochs.
- **Objective:** Maximize area utilization while minimizing congestion and wirelength.

**Results:**

- **Area Utilization:** Improved by **10–12%** over traditional heuristic methods.
- **Layout Realism:** Generated floorplans closely resembled human-designed layouts in terms of module adjacency and clustering of high-fanout nets.
- **Challenges:** Training became unstable for larger designs (>10,000 cells), with occasional infeasible layouts.

## DISCUSSION

### 1. Benefits

Generative AI in VLSI design automation offers:

- **Reduced manual effort**: Engineers spend less time coding HDL or tweaking layouts.
- **Novel solutions**: AI can propose design strategies that humans might overlook.
- **Rapid prototyping**: Designs can be generated and evaluated faster, shortening the design cycle.

### 2. Challenges

Despite its promise, generative AI faces several limitations:

- **Data scarcity**: High-quality labeled datasets for chip layouts and HDL modules are limited.
- **Verification overhead**: Generated designs still require verification and testing.
- **Scalability**: Very large designs may exceed current model capacities.

## FUTURE DIRECTIONS

Potential areas for research include:

- **Self-verifying AI models** that can predict errors in generated HDL.
- **Cross-domain models** that learn from multiple chip architectures.
- **Integration with traditional EDA tools** to provide hybrid human-AI design workflows.
- **Energy-aware generative design**, optimizing not just performance but also power consumption.

## CONCLUSION

Generative AI offers transformative potential for VLSI design automation. By leveraging Transformers, GANs, and RL, designers can automate HDL generation, placement, and layout tasks. While challenges remain in data availability, verification, and scalability, ongoing research demonstrates that AI-assisted design can significantly enhance productivity and innovation in VLSI design. Future work should focus on hybrid methods, self-verifying AI, and better integration with existing EDA flows.

## REFERENCES

1. S. Zhang, et al., "Deep Learning for VLSI Design Automation," *IEEE Trans. CAD*, 2022.
2. L. Chen, et al., "Reinforcement Learning in Chip Placement," *ACM TODAES*, 2021.
3. R. Kumar, et al., "Transformer-based HDL Code Generation," *Journal of Microelectronics*, 2023.
4. P. Singh, et al., "GANs for Floorplan Generation in VLSI," *IEEE Design & Test*, 2021.
5. M. Verma, "AI-driven EDA Tools: A Survey," *Microelectronics Journal*, 2022.
6. J. Roy, et al., "Hybrid AI Methods for ASIC Design," *VLSI Design*, 2023.
7. A. Mehta, et al., "Machine Learning for Timing Analysis," *IEEE TCAD*, 2021.
8. K. Das, "Generative Models in Hardware Design," *Journal of EDA*, 2022.
9. S. Patel, et al., "Reinforcement Learning for Chip Optimization," *Microelectronics*, 2021.
10. R. Sharma, et al., "Data-driven VLSI Design," *International Journal of Electronics*, 2023.

**Cite as:**

Mamta Tiwari, Alok S Pathak, Chandeep Yadav (2026). Generative AI Techniques for VLSI Design Automation. Journal of Research in VLSI Design Tools and Technology, 11(1), 28-38.