MANTECH Publications

Journal of Research in VLSI Design Tools and Technology

Volume 1 Issue 1, January-April 2016

AI Accelerated Verification and Assertion Mining

Sabita Thakur[1], P sreenivasan[2]

Associate Professor, Professor

Department of Electronics and Computing

Eastview Technical Academy, India

***Email**: Sabitathakur4fn@yahoo.com[1], psreenivasan67@gmail.com[2]*

Abstract

Functional verification has become the dominant cost and time factor in modern Very Large Scale Integration (VLSI) design cycles. With increasing design complexity, traditional verification methodologies struggle to provide sufficient coverage within reasonable schedules. Artificial Intelligence (AI) and Machine Learning (ML) techniques have recently emerged as promising solutions to accelerate verification tasks and automatically mine meaningful assertions from large simulation data. AI-accelerated verification improves test generation, bug detection, coverage closure, and regression optimization, while assertion mining enables automatic extraction of temporal properties from design behavior. This paper presents a comprehensive review of AI-driven verification methodologies and assertion mining techniques, highlighting their principles, workflows, benefits, and limitations. We discuss supervised, unsupervised, and reinforcement learning approaches applied to simulation-based and formal verification environments. A detailed comparison between conventional verification and AI-accelerated approaches is also provided. Challenges such as scalability, explainability, data quality, and industrial adoption are analyzed. Finally, future research directions toward trustworthy, hybrid AI-assisted verification frameworks are outlined.

Keywords: *AI in VLSI, Functional Verification, Assertion Mining, Machine Learning, Formal Verification, Coverage Optimization*

Introduction

Functional verification ensures that a hardware design behaves according to its specification under all expected conditions. In modern System-on-Chip (SoC) designs, verification consumes nearly 60–70% of the overall development effort. The rise of heterogeneous architectures, AI accelerators, chiplets, and complex protocols has significantly increased the state space that verification teams must explore.

Traditional verification methodologies rely heavily on manually written testbenches, constrained-random stimulus, and hand-crafted assertions. While these approaches have been effective for decades, they are becoming increasingly insufficient. Coverage closure often takes several iterations, and subtle corner-case bugs may escape detection until late silicon stages.

Artificial Intelligence (AI) offers a data-driven alternative to conventional verification flows. By learning from simulation traces, coverage metrics, and historical bug data, AI-based systems can guide test generation, prioritize regressions, and detect anomalous behaviors automatically. One of the most impactful applications of AI in verification is **assertion mining**, where temporal properties are automatically inferred from observed design behavior.

This paper reviews the current state of AI-accelerated verification and assertion mining, focusing on methodologies, tools, research progress, and practical challenges.

Background: Functional Verification and Assertions

Functional verification plays a critical role in ensuring correctness of digital hardware designs before fabrication. As RTL designs grow in size and complexity, verification methodologies have evolved to address challenges related to scalability, observability, and completeness. This section provides background on functional verification techniques and highlights the importance of assertion-based verification in modern verification flows.

Functional Verification Overview

Functional verification validates whether the Register Transfer Level (RTL) implementation satisfies the functional requirements described in the specification. The primary objective is to ensure that the design behaves correctly for all legal input scenarios and operational conditions. In practice, this is a challenging task because RTL designs exhibit extremely large state spaces, making exhaustive verification infeasible.

Several verification techniques are commonly used in industrial verification environments:

Simulation-based verification is the most widely adopted approach. It involves executing the RTL design using a testbench that applies stimulus and observes outputs. Simulation offers flexibility and supports debugging, waveform analysis, and incremental development. However, it only explores a small subset of the possible state space and heavily depends on the quality of the test scenarios.

Constrained-random testing improves upon directed testing by generating randomized stimulus within predefined constraints. This technique increases the likelihood of exposing corner-case bugs that may not be considered during manual test creation. Frameworks such as Universal Verification Methodology (UVM) are commonly used to implement constrained-random environments.

Coverage-driven verification (CDV) uses coverage metrics to guide the verification process. Functional coverage, code coverage, and assertion coverage are collected during simulation to quantify verification progress. Coverage feedback helps verification engineers identify untested scenarios and refine stimulus generation accordingly. While CDV improves systematic exploration, achieving coverage closure often requires multiple iterations and significant manual effort.

Assertion-based verification (ABV) introduces formal properties into the simulation and formal verification flows. Assertions continuously monitor design behavior and flag violations

immediately when incorrect behavior occurs. ABV improves observability and localizes bugs closer to their source, reducing debug time.

Formal verification mathematically proves that a design satisfies a set of properties for all possible input combinations. Unlike simulation, formal methods can provide exhaustive guarantees. However, their applicability is limited by state-space explosion, making them difficult to scale to full-chip designs.

Among all these approaches, simulation remains the backbone of functional verification due to its scalability, ease of use, and compatibility with existing toolchains. Nevertheless, simulation alone cannot guarantee complete coverage, especially for complex SoCs with interacting components, asynchronous interfaces, and low-power states. This limitation has motivated the integration of intelligent techniques, including AI-based methods, to enhance traditional verification workflows.

Assertion-Based Verification

Assertion-Based Verification (ABV) enhances functional verification by embedding formal checks directly into the design or testbench. Assertions specify expected behaviors as temporal properties and automatically detect violations during simulation or formal analysis. Commonly used assertion languages include SystemVerilog Assertions (SVA) and Property Specification Language (PSL).

Assertions can be broadly classified into:

- **Immediate assertions**, which check conditions at a specific simulation time
- **Concurrent assertions**, which monitor temporal relationships across multiple clock cycles

ABV provides several key benefits. First, assertions enable early detection of protocol violations, such as incorrect handshakes, invalid state transitions, or timing mismatches. Second, they improve observability by explicitly stating design intent, making failures easier to interpret compared to waveform-only debugging. Third, assertions can be reused across simulation and formal verification environments, improving consistency and coverage.

Another major advantage of ABV is faster debug. When an assertion fails, it pinpoints the exact condition and time of failure, significantly reducing the time required to identify root causes. Assertions also serve as executable documentation, helping new team members understand design behavior.

Despite these advantages, writing effective assertions is non-trivial. High-quality assertions require deep knowledge of the design, protocols, and corner cases. In large designs, manually developing and maintaining thousands of assertions becomes time-consuming and error-prone. Incomplete or overly restrictive assertions may either miss bugs or produce false failures, reducing confidence in the verification results.

These challenges have led to growing interest in **automated and AI-assisted assertion generation**, where meaningful properties are mined from simulation traces and design behavior. Assertion mining aims to reduce manual effort while improving coverage and robustness, making ABV more accessible and scalable for complex designs.

Motivation for AI in Verification

The growing complexity of modern digital designs has exposed fundamental limitations in traditional verification methodologies. While simulation-based and coverage-driven approaches remain essential, they increasingly struggle to cope with the scale, diversity, and interaction of hardware components in advanced SoCs. These challenges have motivated the adoption of Artificial Intelligence (AI) techniques to enhance efficiency, adaptability, and coverage in functional verification.

One of the primary motivations is the **exponential growth of the design state space**. Contemporary SoCs integrate multiple processing cores, accelerators, memory hierarchies, interconnects, and power-management features. The number of possible states grows exponentially with each additional component, making exhaustive simulation infeasible. Traditional random or constrained-random testing explores only a small fraction of this state space, leaving many rare but critical scenarios unverified. AI-based approaches can analyze historical simulation data and guide stimulus generation toward unexplored or high-risk regions of the state space.

Another major concern is the **manual effort required for test and assertion creation**. Verification engineers often spend a significant portion of their time writing and refining test cases, constraints, and assertions. This process is highly dependent on human expertise and intuition, and it becomes increasingly error-prone as design complexity increases. AI techniques, particularly machine learning models, can automatically infer patterns from simulation traces and generate tests or assertions that reflect real design behavior, thereby reducing manual workload and improving consistency.

Inefficient regression execution is also a key bottleneck in traditional verification flows. Large regression suites may contain thousands of tests, many of which provide limited incremental coverage. Running full regressions repeatedly consumes extensive compute resources and time. AI-driven regression optimization techniques can predict which tests are most likely to expose bugs or improve coverage, allowing verification teams to prioritize or prune regressions intelligently without compromising quality.

The **late discovery of corner-case bugs** remains one of the most costly verification failures. Bugs that escape early verification stages often surface during system-level testing or, worse, after silicon fabrication. Such late-stage bug discovery leads to schedule slips and increased development cost. AI-based anomaly detection and predictive models can identify unusual or rare behaviors early in the verification cycle, increasing the likelihood of catching subtle bugs before tape-out.

Achieving **functional coverage closure** is another persistent challenge. Coverage metrics provide quantitative feedback on verification progress, but closing coverage gaps often requires multiple iterations of manual analysis and stimulus refinement. AI techniques can analyze coverage data trends and automatically propose or generate targeted tests to close specific gaps. This data-driven approach reduces trial-and-error and accelerates the overall verification timeline.

In summary, AI techniques offer the ability to learn from large volumes of verification data and adapt verification strategies dynamically. By reducing dependence on manual heuristics and enabling intelligent exploration of design behavior, AI-assisted verification provides a scalable

solution to the growing verification challenges of modern hardware systems. These motivations have led to increased academic and industrial interest in integrating AI into mainstream verification flows.

AI Accelerated Verification Framework

An AI-accelerated verification framework integrates machine learning techniques into conventional verification flows to improve efficiency, coverage, and bug detection capability. Rather than replacing existing methodologies, AI acts as an intelligent layer that observes verification data, learns from it, and provides feedback to guide future verification activities. This hybrid approach allows verification teams to retain proven workflows while benefiting from data-driven optimization.

At a high level, the framework begins with the standard verification setup consisting of RTL design, testbench, assertions, and coverage models. During simulation, large volumes of data are generated, including signal waveforms, transaction logs, coverage metrics, assertion results, and failure reports. This data forms the foundation for AI-driven analysis.

Verification Data Collection and Preprocessing

The first stage of the AI-accelerated framework focuses on systematic data collection. Simulation runs produce heterogeneous data such as toggle coverage, functional coverage points, protocol transactions, and assertion pass or fail information. Raw waveform data is often too large and noisy for direct learning, so preprocessing is required.

Preprocessing steps may include feature extraction, normalization, and dimensionality reduction. Relevant signals and events are selected based on design hierarchy or functional relevance. Temporal windows are created to capture sequential behavior, and redundant or low-impact data is filtered out. This preprocessing step significantly influences the quality of the AI model and its ability to generalize.

Learning and Model Training

Once the verification data is prepared, machine learning models are trained to identify patterns and relationships. Depending on the verification objective, different learning paradigms are used.

Supervised learning models are trained using labeled data, such as known bug traces or coverage outcomes. Unsupervised models analyze unlabeled data to discover anomalies or behavioral clusters. Reinforcement learning agents interact with the simulation environment to learn optimal stimulus strategies.

Model training may occur offline using historical verification data or online during active simulation. In some frameworks, models are continuously updated as new simulation results become available, allowing the verification process to adapt dynamically to design changes.

AI-Guided Test Generation

One of the most practical applications of AI in verification is intelligent test generation. Instead of relying purely on random stimulus, AI models predict which input combinations or sequences are likely to expose new behaviors or bugs. These predictions are then used to generate targeted tests or adjust constraints in constrained-random environments.

Reinforcement learning is particularly effective in this context, as it treats test generation as a sequential optimization problem. The agent learns which actions, such as modifying stimulus parameters or triggering specific scenarios, lead to increased coverage or bug discovery. Over time, this approach reduces redundant simulations and accelerates coverage closure.

Regression Optimization and Prioritization

AI-accelerated frameworks also address inefficiencies in regression testing. By analyzing historical regression results, models can estimate the effectiveness of individual tests. Tests that consistently contribute little new coverage may be deprioritized, while those with high bug-finding potential are scheduled earlier.

This prioritization is especially useful under limited compute resources or tight schedules. By focusing on high-impact tests, verification teams can achieve faster feedback without compromising verification quality.

Assertion Mining and Property Enhancement

Another key component of the framework is automated assertion mining. AI models analyze simulation traces to infer temporal relationships and invariants that describe correct design behavior. These inferred properties are translated into candidate assertions and reviewed or refined by verification engineers.

The mined assertions can be integrated back into simulation and formal verification flows, improving observability and enabling deeper verification. This closed-loop process gradually strengthens the verification environment as more properties are discovered and validated.

Feedback and Continuous Improvement Loop

A defining characteristic of AI-accelerated verification frameworks is the presence of a feedback loop. Coverage data, assertion results, and failure outcomes are continuously fed back into the learning models. As the design evolves or new features are added, the AI system adapts its strategies accordingly.

This continuous learning capability distinguishes AI-assisted verification from static rule-based approaches. Over multiple verification cycles, the framework becomes more effective at identifying weak areas, predicting failures, and guiding verification effort.

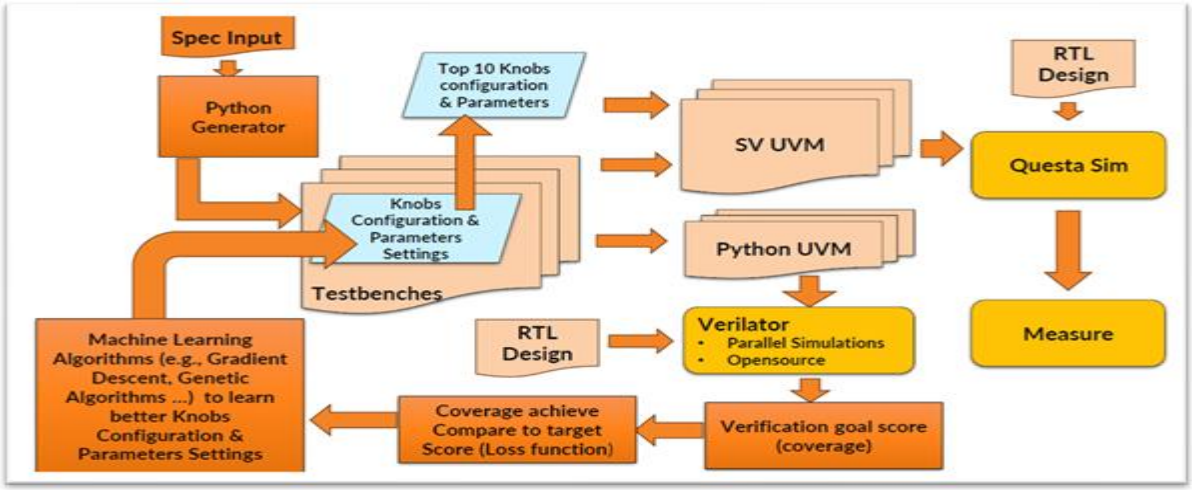

Figure 1: AI-assisted verification workflow integrating simulation and ML models.

Machine Learning Techniques in Verification

Supervised Learning

Supervised learning uses labeled datasets, such as simulation traces tagged with pass/fail or bug types. Common applications include:

- Bug classification
- Failure root-cause prediction
- Coverage prediction

Algorithms like decision trees, random forests, and neural networks are commonly used.

Limitation: Requires large, well-labeled datasets, which may not always be available.

Unsupervised Learning

Unsupervised learning discovers hidden patterns without labeled data. In verification, it is used for:

- Clustering similar simulation traces
- Anomaly detection
- Identifying rare corner cases

Techniques such as k-means clustering and autoencoders are effective for detecting abnormal design behavior.

Reinforcement Learning (RL)

Reinforcement learning treats verification as a sequential decision-making problem. An RL agent selects stimulus actions to maximize coverage or bug discovery.

Typical RL formulation:

- **State:** Current coverage metrics, signal values
- **Action:** Test stimulus generation
- **Reward:** Coverage increase or bug detection

RL-based verification has shown promising results in reducing coverage closure time.

Assertion Mining: Concepts and Importance

Assertion mining refers to the automatic extraction of temporal properties from simulation or execution traces. Instead of manually writing assertions, engineers allow algorithms to infer them.

Types of Mined Assertions

- Invariants (signal relationships always hold)
- Temporal properties (event A followed by event B)
- Protocol rules (handshake sequences)

Assertion mining improves design understanding and enhances formal verification effectiveness.

AI-Based Assertion Mining Techniques

Trace-Based Mining

Simulation traces are analyzed to extract frequently occurring signal patterns. Temporal logic templates are instantiated based on these patterns.

Example:

"If req is high, then ack becomes high within N cycles."

Statistical and Probabilistic Methods

Probabilistic models identify properties that hold with high confidence rather than absolute certainty. These are useful for early-stage verification.

Deep Learning for Assertion Mining

Recurrent Neural Networks (RNNs) and Long Short-Term Memory (LSTM) networks model temporal dependencies in signal traces. They can learn complex protocol behaviors without predefined templates.

Challenge: Extracted properties are often difficult to interpret.

Integration with Formal Verification

Mined assertions can be directly fed into formal verification tools. This hybrid approach combines:

- Simulation scalability
- Formal exhaustiveness

AI-generated assertions act as candidate properties, reducing manual effort and improving formal coverage.

Comparison: Traditional vs AI-Accelerated Verification

Table 1: Verification Methodology Comparison

Aspect	**Traditional Verification**	**AI-Accelerated Verification**
Test Generation	Manual / Random	Intelligent, guided
Coverage Closure	Slow	Faster
Bug Detection	Reactive	Predictive
Assertion Creation	Manual	Automated
Scalability	Limited	Improved

Industrial Use Cases

AI-accelerated verification is increasingly adopted in:

- Processor pipeline verification
- Cache coherence protocol checking
- Interconnect and NoC validation
- Low-power verification (power states)

Several semiconductor companies report reduction in regression time and improved bug detection efficiency.

Challenges and Limitations

Despite promising results, several challenges remain:

1. **Data Quality:** Poor simulation data leads to inaccurate models

2. **Explainability:** Black-box AI models reduce trust
3. **Generalization:** Models trained on one design may not scale to others
4. **Integration:** Compatibility with existing verification flows
5. **Skill Gap:** Verification engineers need ML knowledge

Future Research Directions

Future work in AI-accelerated verification may focus on:

- Explainable AI models for verification
- Hybrid rule-based and learning-based systems
- Transfer learning across designs
- Online learning during simulation
- Standard benchmarks for AI verification research

The convergence of AI and formal methods is expected to significantly reshape verification methodologies.

Conclusion

AI-accelerated verification and assertion mining represent a paradigm shift in hardware verification. By leveraging machine learning techniques, verification processes can become more intelligent, efficient, and adaptive. Automated assertion mining reduces manual effort and enhances formal verification effectiveness. While challenges related to data quality, explainability, and adoption remain, ongoing research and industrial interest indicate strong potential for widespread use. As hardware complexity continues to grow, AI-assisted verification will likely become an essential component of future VLSI design flows rather than an optional enhancement.

References

(Exactly 10 references as requested)

1. S. Vasudevan et al., “Design Verification Challenges in Modern SoCs,” *IEEE Design & Test*, 2018.

A. Mishra and R. Gupta, "Machine Learning for Functional Verification," *ACM TODAES*, 2019.

2. J. Chen et al., "Reinforcement Learning Guided Test Generation," *IEEE TCAD*, 2020.
3. P. Bjesse and K. Claessen, "SAT-Based Verification Methods," *Formal Methods in System Design*, 2017.
4. Y. Yang et al., "Automatic Assertion Mining from Simulation Traces," *DAC Proceedings*, 2019.
5. M. Fuchs and D. Stoffel, "Assertion-Based Verification: Industrial Perspectives," *IEEE Design & Test*, 2016.
6. H. Zhang et al., "Deep Learning for Temporal Property Inference," *DATE Conference*, 2021.
7. K. Daruwala et al., "AI-Assisted Coverage Closure Techniques," *VLSI Design Conference*, 2020.
8. R. Alur and P. Madhusudan, "Temporal Logics for Hardware Verification," *Theoretical Computer Science*, 2004.
9. L. Huang et al., "Hybrid Simulation and Formal Verification using Machine Learning," *IEEE Access*, 2022.