Journal of Information Systems Engineering and Management
2026, 11(1s) e-ISSN:
2468-4376
https://www.jisem-journal.com/ **Research Article**

Closed-Loop EDA Verification with AI: Autonomous Planning for Adaptive Simulation

Praveen Kumar Manchikoni Surendra
Central Michigan University, USA

ARTICLE INFO

Received: 08 Jan 2026
Revised: 12 Jan 2026

ABSTRACT

The area of EDA verification faces continuous issues with static regression testing methods that are not adaptable to new emerging areas of coverage or unexpected fail points in testing cycles. Conventional chip design verification methodologies run pre-defined test suites without reactivity to interim results in real time, leading to suboptimal allocation of resources and slowed progress of coverage achievement. Autonomous planning tools based on chain of thought reasoning and reinforcement learning bring revolutionary potential to dynamic verification management. These cognitive planners are fully responsive to simulation results continuously and can dynamically adjust test execution plans based on interim outcomes. The closed-loop system allows directed test stimuli aimed at particular areas of uncovered functionality while keeping necessary testing baselines. Upon failure points, planners apply concentrated debugging flows and temporarily shift allocation of resources for root cause investigation. Detailed decision logging creates record linkage of autonomous actions triggered by core evidence in all phases and steps of chip design verification testing. Architecturally neutral tool development enables easy interconnection of various simulators and platforms of coverage on standardized interfaces. The dynamically adaptive paradigm of verification holds a great promise of accelerating convergence of testing coverage, improving allocation of computing resources on simulators, and decreasing human intervention points in integrated circuit debugging.

Keywords: Autonomous Chip Design Verification Planning, Adaptive Simulation Orchestration, Closed-Loop Feedback Mechanisms, Chain-Of-Thought Reasoning, Coverage-Driven Test Generation In Semiconductor Chip Design

1. Introduction to Verification Challenges in Modern Electronic Design Automation

The current complexity of integrated circuit designs has raised the level of verification tasks to unprecedented heights, where current system-on-chip designs involve billions of transistors and highly complex functional dependencies. The current verification processes heavily depend on regression test suites, where specific sets of test cases are performed in a predetermined order to check the functionality of the designs after every new version of the design. Industry-wide analysis indicates that verification tasks remain a major project consumer in terms of project time and resource utilization, constituting a major bottleneck in the development flow of semiconductors. The static nature of current regression test suites inherently involves limitations in coping with dynamic issues that arise in the process of verification, where the specific sets of tests run irrespective of the progress and coverage necessities [1].

Regression suites within the traditional verification process do not change unless altered by verification engineers. If simulation results indicate areas of test coverage or unexpected errors in designs, the verification team has to finish the current regression cycles before addressing these issues. This serves as an inefficiency accumulator, as the verification engineer responds to test coverage results in a non-

Journal of Information Systems Engineering and Management
2026, 11(1s) e-ISSN:
2468-4376
https://www.jisem-journal.com/ Research Article

immediate fashion, analyzes the deficit regions, develops test patterns independently, and reapplies the regression suites after finishing the preparatory work.

The stiffness of traditional approaches in static regression can be particularly challenging with increasing complexity and rising exponential verification state spaces. Today, common verification settings often involve situations where some aspects of a design may be inadequately covered until late stages, leading to rushed test development efforts. Industrial analyses conducted on metrics for a wide array of verification projects reveal some challenges in obtaining thorough coverage, with considerable efforts going into addressing functional holes during the final stages of a project. That said, the industry understands that coverage metrics do not give a complete picture regarding the adequacy of a verification process, but these metrics remain critical for the assessment process [2].

The gap between computing power and planning complexity introduces bottlenecks that impede the efficient usage of verification capabilities. While the environment for executing simulations over large data distributions for computing has improved, incorporating vast parallel computing capabilities for large computing clusters, the verification planning processes have continued with traditional approaches, which depend on human decisions in the selection and prioritization of verification tests. The gap in these verification processes, which overlooks the opportunity for efficiency gains, exists because the verification processes involve large computing capabilities. The verification community has realized the necessity for more intelligent verification management solutions that can dynamically adapt to new verification requirements, based on their reaction to verification outcomes, and make self-directed decisions for test strategies.

2. Fundamentals of Autonomous Planning Systems in Verification

Autonomous planning systems are a type of artificial intelligence architecture that is used for developing sequences of actions that are required for transitioning from initial states to goal states while abiding by given constraints. Recent developments in language models have brought about new approaches for planning using chain-of-thought reasoning, in which models are required to perform reasoning before arriving at their decision or judgment concerning a particular query being asked. Studies done on research work involving prompting language models have revealed that taking into consideration step-by-step reasoning in language models significantly boosts their performance for complex queries involving multiple logical steps for obtaining a resultant judgement [3].

In verification planning settings, chain-of-thought mechanisms allow language models to provide testing rationale based on test selection choices, taking into consideration the current coverage values, test performance history, and the available resources in a systematic way. In testing rationale application to verification orchestration, the models can provide justifications on why particular test choices are recommended, why particular coverage needs attention, and the relation of proposed actions to overall verification objectives in a structured setting. These particularities of testing rationale provide verification engineers with the ability to interpret the rationale of the automated decision-making process of the verification orchestration tools and provide a manual override option as and when needed. Machine learning techniques and applications in testing and optimization have received growing interest as a result of the complexity of verification, surpassing the capabilities of manual handling.

Systematic studies on machine learning integration in automated testing have brought various applications, ranging from test input, test prioritization, fault localization, and oracle implementation. Literatures on machine learning applications in testing and optimization has various approaches directed at different conditions, ranging from supervised machine learning based on experience using testing data, unsupervised machine learning that involves data mining and design patterns, and other

Journal of Information Systems Engineering and Management
2026, 11(1s) e-ISSN:
2468-4376
https://www.jisem-journal.com/ Research Article

approaches based on machinelearning using reinforcement techniques directed at experimenting on optimization during verification processes, maximizing the benefit of testing at various levels [4].

The incorporation of planning strategies with the verification infrastructure entails keen consideration for the interface and computational complexity. The planning agent needs to interact with the realtime coverage information produced by the simulation environments, the design specification, and the commands for the test generators and simulation orchestration systems. The computational expense in the planning decisions is an essential constraint since the planning expense has to be in line with the simulation running time for the applicability of an adaptive model. The planning system has to cope with the uncertainties in the simulation results and the measurements for the probabilistic nature in dealing with the stochastic process in the production of randomly generated tests and limitations in the coverage metrics.

Planning Approach	Reasoning Mechanism	Application Domain	Integration Complexity	Decision Transparency
Chain-of-Thought	Step-by-step logical inference	Test selection and prioritization	Moderate	High
Supervised Learning	Historical test data patterns	Fault localization and oracle automation	Low	Moderate
Unsupervised Learning	Design behavior pattern discovery	Test input generation	Moderate	Low
Reinforcement Learning	Iterative experimentation	Sequential decision optimization	High	Moderate

Table 1: Performance Comparison of Planning Paradigms in Verification Systems [3, 4]

3. Architecture of Adaptive Simulation Orchestration Systems

The architecture of an adaptive simulation orchestration environment is built from multiple interacting elements that altogether contribute to automatic control of verification campaigns. At its foundation is the monitoring subsystem, which constantly acquires information from executing simulation threads concerning functionality coverage information, code coverage information, assertion firing rates, and error detection capability. Coverage information is considered a basic tool used for monitoring verification progress and coverage gaps within validation efforts. Investigations into research using coverage information within hardware verification illustrate that appropriate coverage information evaluation depends on metric selection according to verification goals because different categories of information offer diverse insights into design exercise coverage [5].

The monitoring infrastructure is compatible with conventional verification tools, interpreting either code or simulation data to indicate the level of simulation achieved. This is possible by using functional or code coverage, which can indicate whether particular design elements, state space, and cross-product points have been simulated. The hardware design code can be measured by code coverage; however, this still offers two ways of ascertaining the level of simulation achieved. The monitoring system has minimal latency, ensuring that any decisions made about simulation planning take into account the present simulation status rather than being influenced by previous simulation levels in early stages of regression simulation.

The resource allocation system controls how computational resources are allocated for competing verification tasks based on their priority levels, which are defined by the planning agent. Typical

Journal of Information Systems Engineering and Management
2026, 11(1s) e-ISSN:
2468-4376
https://www.jisem-journal.com/ Research Article

verification environments involve compute clusters that have large computing capabilities, while the computational requirements for a particular simulation change based on design complexity as well as the level of parallelism used in the simulations. The allocation system monitors the resource levels for immediate verification requirements, which entails monitoring lists for test executions that are yet to be done. The allocation system can therefore dynamically change the simulation scheduling based on urgent verification requirements.

Prioritize protocols assist in deciding on which pre-existing tests should be considered for a physical execution and how they should be performed next. There is authoritative research carried out on prioritizing test cases, which proved that optimized selection of regression tests may result in fulfilling fault detection objectives effectively and faster than random selection patterns. Analysis on varied prioritization approaches proved that optimized coverage selection approaches outperform random and requirements selection approaches since tests that target uncovered segments of code and functions have a faster fault detection capability compared to others [6].

Metric Category	Measurement Target	Verification Perspective	Latency Level	Resource Intensity
Functional Coverage	Design features and state transitions	Verification intent achievement	Minimal	Moderate
Code Coverage	HDL statements and branches	Structural design exercise	Minimal	Low
Assertion Firing Rates	Protocol compliance checks	Correctness property validation	Real-time	Low
Cross-Product Coverage	Feature interaction combinations	Complex scenario exercise	Minimal	High

Table 2: Coverage Metric Categories and Monitoring Capabilities in Adaptive Systems [5, 6]

4. Closed-Loop Feedback Mechanisms for Coverage Optimization

The use of closed-loop feedback mechanisms facilitates improvements to the verification strategies through the accumulation of evidence obtained from the results of simulations. Coverage analysis is one aspect that plays a fundamental role in the process of functional verification, allowing the evaluation of the completeness of the verification process and identifying the regions that need further focus during the testing process. Frameworks that deal with the measurement and analysis of coverage offer strategies that enable the creation of coverage models, the collection of coverage information during the execution of simulations, and the focus during the process of generating tests for the untested regions [7].

Coverage hole identification algorithms assess functional coverage information for the purpose of identifying certain features of designs, transition coverage, or cross-product combination coverage that are yet to be tested by the test suites. Coverage identification involves the evaluation of coverage databases in search of items with zero or low hit counts, which denote a lack of attention by test suites. Advanced coverage identification involves the examination of temporal trends of coverage items whose hit rates are gradually declining over regression cycles or features resistant to coverage identification after repeated test generations. The coverage analysis component supports coverage databases that register the evolution of coverage information features over time, which can then identify deviant trends of previously covered items becoming uncovered after modifications in the design.

Journal of Information Systems Engineering and Management
2026, 11(1s) e-ISSN:
2468-4376
https://www.jisem-journal.com/ Research Article

Upon finding deficiency areas in coverages, the system initiates test generation tasks for specific functionalities yet to be covered. Constraint-driven random test data generation has recently been found useful for designing tests that stimulate targeted design activities with the randomization advantage maintained. Solvers and test data generation have been combined in test generation systems to process declarative constraints defining test generation objectives. These declarative constraints define test generation objectives, and solvers help process the complexity associated with generating test data that satisfies these objectives. This declarative specification of constraints allows verification engineers to define test objectives at highly abstract levels, independent of the complexity associated with test data generation [8].

The modification of test suites through adaptive testing during active verification cycles is a paradigm shift relative to traditional regression methods through static models. With the identification of highvalue test additions or the realization of reduced returns by some of the existing tests by the planning agent, it updates active test queues without waiting for a cycle to finish. This results in timely reactions to verification demands since new tests can begin execution immediately after identifying gaps in verification. Validation methods of the incremental kind oversee the addition of newly derived tests within regression domains. Initial validation cycles are executed through scope-limited setups to confirm functionality before embarking on comprehensive testing.

Feedback Component	Primary Function	Analysis Technique	Response Time	Adaptation Capability
Coverage Hole Identification	Gap detection	Database evaluation with trend analysis	Minutes	High
Constraint-Based Generation	Targeted test creation	Declarative constraint solving	Minutes to Hours	Moderate
Adaptive Suite Modification	Dynamic test queue updates	Value assessment algorithms	Real-time	High
Incremental Validation	New test verification	Limited scope execution	Minutes	Moderate

Table 4: Closed-Loop Feedback Components and Operational Characteristics [7, 8]

5. Failure Response and Diagnostic Test Deployment

The automated mechanisms for detecting anomalies in the simulation results and outputs are continually checking for irregular results in the simulation outputs. The system for detection analyzes the simulation results and applies statistical methodologies for identifying irregular events in the simulation results. In contemporary verification environments for tests, there is a considerable number of assertions in the testbench, each tracking different properties of verification, including the interface protocol and the expected data consistency. When there is an occurrence in the results of the verification tests, quick pathologies for detection and diagnosis could help in avoiding stalled progress. The techniques for debugging in traditional methods involve thorough checks and hypothesis testing.

Automated fault injection approaches can be used to produce data sets that help identify system behaviors and train models for diagnosis. Research on fault injection approaches as part of failure prediction studies has found that injecting faults leads to system behaviors and manifestations of system failures useful for system robustness analysis and developing predictive models. System behaviors and system manifestations derived through the fault injection method help identify relationships in fault

Journal of Information Systems Engineering and Management
2026, 11(1s) e-ISSN:
2468-4376
https://www.jisem-journal.com/ Research Article

data and system manifestations used as part of diagnoses in case of unforeseen system failures during regular system analysis and testing processes [9].

Once important anomalies are recognized, planning agents launch targeted generations of the diagnostic sequence to identify root causes of failure that can be isolated. Test sequences for diagnostics involve the use of varying conditions to limit the range of design that influences failure triggers based on observed results. Sequence generation involves initial analysis of test failures to consider variable elements, including data inputs, timing, and initial state conditions. Test sequences are then carried out, varying most elements while varying others considered to influence failures based on suspected elements that influence failure conditions. All these work similarly to software debugging that involves searches to identify problematic areas to address properly. Cause reduction techniques offer optimal ways of simplifying failing test cases and, at the same time, retaining failurecausing properties. Delta-debugging algorithms start eliminating parts of the test data or program execution paths and then examine if the minimized test cases can still show failure behaviors. By an optimal process of elimination, the algorithms find the minimal test cases demonstrating failure behaviors through the shortest input sequences and design settings. The minimal test cases significantly improve designer debugging by concentrating attention on the critical failure-causing aspects while discarding unnecessarily complex information. It has also been proven that cause reduction is feasible, even when the initial failure is not easy to repeat or debug [10].

Diagnostic Technique	Failure Detection Method	Isolation Strategy	Complexity Reduction	Reproducibility Support
Automated Anomaly Detection	Pattern matching and statistical analysis	Assertion monitoring	Moderate	High
Fault Injection	Controlled perturbation	Correlation analysis	Low	High
Diagnostic Sequence Generation	Systematic input variation	Binary search approach	High	Moderate
Delta Debugging	Iterative test minimization	Incremental reduction	Very High	Moderate

Table 4: Diagnostic Techniques for Failure Isolation and Root Cause Analysis [9, 10]

6. Traceability, Reproducibility, and Tool-Agnostic Integration

Decision logging structures enable the capture of thorough records of planning agent decisions, justification, and conditions for activation during the course of verification campaigns. Every planning agent decision results in the creation of entries in the logging structure for the capturement of the state of verification at the instant of the decision, including the present status of verification coverage, recently completed tests, resource usage levels, and verified gaps and failures in coverage. The logging structure records distinct actions relating to the employment of tests, changes in their priority levels, and resource assignments.

Evidence-based audit trails create direct connections between planning decisions and the verification evidence that justifies the decisions. With the addition of directed tests to close the gaps by planning agents, evidence-based audit trails capture the exact items that went unprotected, the coverage history, and the justification for choosing specific generation strategies. In the context of the use of diagnostic

sequences to counteract failures, the inclusion of information about the symptoms, messages, and affected test sequences by evidence-based audit trails facilitates the understanding and verification of decisions made by planning agents to trust autonomous planning strategies. Regression testing methods for evolving software systems serve as relevant antecedents for preserving the connection between the execution and modification activities. Studies related to selective regression testing defined strategies to determine susceptible regression tests to software change, codified the dependencies between code and test cases, and documented justifications for selecting specific strategies [11].

Generic interface design facilitates the integration of various simulation and coverage analysis tools without the need for adapting the agents for particular vendor platforms. The generic interface design specifies abstract operations such as the invocation of simulation, obtaining coverage information, requesting test generation, and querying the result, which are realized by vendor platform-specific adapter modules. The design protects the planning agents from implementation details such as the particular commands for invoking the tool, file format, and representation schemes. Black box verification methods, which check system behaviors based on external observation and do not involve implementation details, are apt architectural examples for generic integration. These methods specify architectural interaction protocols for verification systems to operate on components with welldefined interfaces independent of implementations [12].

The challenge of proving reliability in self-governing agents in production environments presents an actual need for evaluating objectives in design tasks, involving planning capability, resource use, and especially effective verification. Methods of validation include assessing managed verification explorations by self-governing agents against unmanaged exploration using static regression. This enables an architecture independent of tools, being beneficial to industries in implementing selfgoverning planning while not wasting investments in traditional verification infrastructure.

Conclusion

Autonomy in planning systems embodies a revolution in the process of electronic design automation in chip design verification, shifting the focus from traditional static regression planning to adaptive planning systems that respond to real-time simulation results in modern chip designs. The combination of chain of thought explanation and reinforcement learning allows intelligent systems to react to real-time simulation results by adapting how tests are performed, utilizing compute resources efficiently, and developing directional tests for areas of identified coverage holes. Closed-loop feedback systems are used to improve the adaptation of verification continuously flows through accumulating evidence, while automated failure response plans utilize diagnostic steps for efficient root cause identification. Comprehensive decision recording and evidence-track audit trails for traceability throughout the verification process enable regulatory compliance and analysis of the entire process flow after campaign execution. Architecture for the autonomous verification system has been developed in a tool-independent framework for easy integration with existing infrastructure for verification without creating obstacles in adaptation and integration. The autonomous verification system has great potential in terms of accelerating the timeline for coverage closure, reducing compute utilization costs in distributed compute systems, and minimizing human intervention in the process of test selection and priority assignment. The increasing complexity of the integrated circuit raises the importance of autonomous system solutions using AI for maintaining positive quality for verification while addressing compute costs and timeline limitations.

References

[1] Siemens, " 2020 Wilson Research Group functional verification study," [Online]. Available:

https://resources.sw.siemens.com/en-US/white-paper-2020-wilson-research-group-functionalverification-study/

[2] Andreas Meyer and Harry Foster, "Metrics in SoC Verification," [Online]. Available: https://dvcon-proceedings.org/wp-content/uploads/metrics-in-soc-verification.pdf

[3] Jason Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," arXiv:2201.11903, 2023. [Online]. Available: https://arxiv.org/abs/2201.11903

[4] Afonso Fontes and Gregory Gay, "The integration of machine learning into automated test generation: A systematic mapping study," Wiley, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.1002/stvr.1845

[5] S. Tasiran and K. Keutzer, "Coverage metrics for functional validation of hardware designs," IEEE Design & Test of Computers, Volume 18, Issue 4, 2001. [Online]. Available: https://ieeexplore.ieee.org/document/936247

[6] S. Elbaum et al., "Test case prioritization: A family of empirical studies," IEEE Transactions on Software Engineering, Volume 28, Issue 2, 2002. [Online]. Available: https://ieeexplore.ieee.org/document/988497

[7] Andrew Piziali, "Functional Verification Coverage Measurement and Analysis," Springer, 2008. [Online]. Available: https://link.springer.com/book/10.1007/b117979

[8] Yehuda Naveh et al., "Constraint-based random stimuli generation for hardware verification," IAAI'06: Proceedings of the 18th conference on Innovative applications of artificial intelligence, Volume 2, 2006. [Online]. Available: https://dl.acm.org/doi/10.5555/1597122.1597129

[9] João R. Campos and Ernesto Costa, "Fault Injection to Generate Failure Data for Failure Prediction: A Case Study," IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), 2020. [Online]. Available: https://ieeexplore.ieee.org/document/9251077

[10] Alex Groce et al., "Cause reduction: delta debugging, even without bugs," Software Testing, Verification & Reliability, Volume 26, Issue 1, 2016. [Online]. Available: https://dl.acm.org/doi/10.1002/stvr.1574

[11] Mary Jean Harrold et al., "Regression test selection for Java software," ACM SIGPLAN Notices, Volume 36, Issue 11, 2001. [Online]. Available: https://dl.acm.org/doi/10.1145/504311.504305

[12] DORON PELED et al., "BLACK BOX CHECKING," [Online]. Available: https://www.cs.rice.edu/~vardi/papers/pstv99.pdf