International Journal of Latest Engineering and Management Research (IJLEMR)
ISSN: 2455-4847
www.ijlemr.com || Volume 10 – Issue 02 || February 2025 || PP. 69-76

Optimizing Semiconductor Testing: Leveraging Stuck-At Fault Models for Efficient Fault Coverage

Vijayaprabhuvel Rajavel
Semiconductor Design & Test Specialist, California, USA

Abstract: With the increasing scale of computing systems and the growing complexity of modern processor architectures, the importance of in-field testing mechanisms continues to rise. Traditional scan methods and automatic test pattern generation (ATPG) are often too resource-intensive or ineffective when adapting to long test programs and out-of-order execution cores. This study proposes an automated Software-Based Self-Testing (SBST) method for RISC-V cores based on reinforcement learning (RL). The key contribution lies in the use of toggle coverage as a proxy metric, significantly reducing computational overhead during RL agent training while maintaining a high level of stuck-at fault detection. Experimental results on synthesized RISC-V cores (both in-order and out-of-order) demonstrate that the proposed RL-SBST approach achieves over 90% stuck-at fault coverage with relatively short test programs (approximately 200 instructions), significantly outperforming random strategies and RISCV-DV-based techniques (an automated test generation method that combines features of the RISC-V architecture with dynamic fault dependency models for optimized and efficient defect coverage). This study highlights the potential for integrating RL algorithms and proxy metrics to optimize in-field testing of computing devices without requiring significant hardware modifications or increasing system downtime. The findings are relevant to researchers and practitioners in microelectronics, automated testing, and fault modeling.

Keywords: software-based self-testing, stuck-at faults, reinforcement learning, toggle coverage, in-field testing, test programs, RISC-V.

Literature Survey

Modern computing systems are increasingly affected by the occurrence of faults during post-deployment operation. According to recent reports, major IT companies such as Meta and Google encounter a low but significant failure rate of processor cores in their server farms, which can lead to silent data corruption (SDC) and, consequently, unpredictable consequences for critical applications [4]. The emergence of such faults is attributed to the increasing complexity of processor microarchitectures and the growing limitations of nanometer-scale manufacturing—smaller transistors are more prone to degradation. Factory-level testing does not always fully eliminate the risk of permanent (e.g., stuck-at) and recurring defects that may manifest during operation [1].

Given that, in addition to transient faults (soft errors), permanent faults are becoming a critical concern, particularly in large-scale systems, the demand for effective real-time diagnostic solutions is increasing. Traditional scan chain methods using external automatic test equipment (ATE) are too resource-intensive for in-field testing, as they require halting the core's operation, accessing the scan chain, and significantly increasing system downtime. In contrast, software-based self-testing (SBST) enables diagnostics "at speed" without costly hardware intervention by generating test instruction sequences executed directly by the target processor. However, the challenge of automatically generating such programs with high fault coverage while maintaining a reasonable test length remains unresolved [1].

Thus, the comprehensive problem of detecting hardware faults in the field for modern processor cores remains only partially addressed and requires further investigation.

J. Seo and H. Cho [1] propose a method in which reinforcement learning algorithms are used to generate effective instruction sequences for SBST of processor cores. In the same paradigm, C. Y. Chen and J. L. Huang [2] formulate the hypothesis that reinforcement learning can enhance test coverage efficiency by optimizing command sequences, while N. Pfeifer et al. [9] demonstrate the application of this approach for targeted verification test generation for shared memory resources, indicating a methodological shift from static testing schemes to dynamic, self-learning systems.

Zhang Y. et al. [3] describe a method for generating SBST using bounded model checking (BMC), focusing for the first time on non-functional superscalar processors (OOE). Zhang Y. et al. [7] introduce a temperature-aware approach that considers the impact of high temperatures on detected faults, addressing a research gap related to the insufficient evaluation of thermal effects in traditional testing schemes.

Deligiannis N. I. et al. [6] propose an automated approach based on solving MaxSAT problems for generating programs that facilitate repeatable and consistent switching activity, significantly increasing the likelihood of detecting stuck-at faults. In addition, Deligiannis N. I., Cantoro R., and Reorda M. S. [8] demonstrate the application of evolutionary algorithms to optimize the distribution of switching activity across different processor core modules, which not only enhances testing efficiency but also enables a detailed examination of error distribution patterns within the system.

Kochte M. A. and Wunderlich H. J. [10] emphasize the development of comprehensive self-diagnostic systems capable of responding to dynamic functional changes, which they consider a key element in improving the reliability of modern microprocessor systems.

Dixit H. D. et al. [4] investigate the phenomenon of silent data corruption in large-scale systems, highlighting an existing research gap in accounting for the exponential growth of computing systems and its impact on test accuracy. Similarly, Hochschild P. H. et al. [5] examine the issue of insufficient test coverage for processor cores, where undetected faults may persist, necessitating a reassessment of traditional testing approaches amid increasing system complexity.

The research gap lies in the unresolved issue of automating SBST program generation. Despite the active use of Bounded Model Checking(BMC), RL, and evolutionary algorithms, there is still no unified approach capable of scaling to large cores while achieving high fault coverage without excessive computational overhead.

The objective of the study is to explore the optimization of semiconductor testing by leveraging fault activation models for efficient fault coverage.

The scientific novelty lies in the extensive review and analysis of optimization strategies for semiconductor testing using fault activation models. Based on the conducted analysis, recommendations have been formulated for the optimal application of these models.

The hypothesis suggests that using toggle coverage as a proxy metric in reinforcement learning can achieve comparably high (or even higher) levels of stuck-at fault detection compared to direct yet computationally expensive stuck-at fault coverage calculation.

The methodology is based on a comparative analysis of scientific publications.

Research Methods

Effective testing of digital devices requires that the maximum possible number of circuit nodes be activated, meaning they switch between different states, and that the results of such activation are captured at the outputs. Traditionally, test design follows two principles [1]:

Fault activation: generating a set of input vectors that cause the target circuit node to assume an erroneous value if faulty (e.g., stuck-at-0 or stuck-at-1, shorted to a neighboring signal, etc.).

Propagation to output: ensuring that an error occurring in an internal node is reflected in the final observable architectural state, such as a general-purpose register in a processor or memory.

For simple combinational circuits, formal methods such as automatic test pattern generation (ATPG) have long been used. However, for complex microprocessors with multi-stage pipelined and/or superscalar architectures, traditional ATPG faces challenges due to the exponential state growth. Consequently, software-based self-testing (SBST) approaches are becoming increasingly relevant, as they can be applied to an operational device [1].

There are several primary fault models, each corresponding to a specific type of physical and technological defect (Fig. 1).

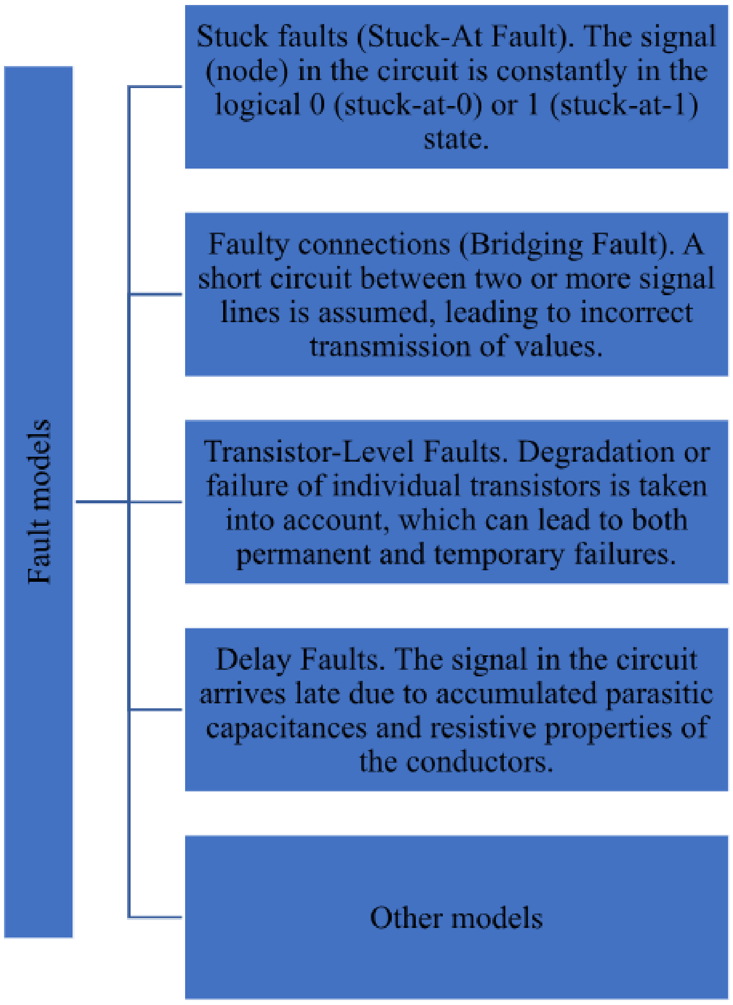
Fault models
Stuck faults (Stuck-At Fault). The signal (node) in the circuit is constantly in the logical 0 (stuck-at-0) or 1 (stuck-at-1) state.
Faulty connections (Bridging Fault). A short circuit between two or more signal lines is assumed, leading to incorrect transmission of values.
Transistor-Level Faults. Degradation or failure of individual transistors is taken into account, which can lead to both permanent and temporary failures.
Delay Faults. The signal in the circuit arrives late due to accumulated parasitic capacitances and resistive properties of the conductors.
Other models

Fig.1. Fault models [1].

This study focuses primarily on stuck-at faults; however, certain aspects of the discussion, such as the switching activity mechanism, can be extrapolated to other types of defects as well [1, 3] (Table 1).

Table 1. Main fault models and their brief description [1].

Fault Model	**Description**	**Applicability**
Stuck-At Faults	A node or line remains stuck at logical 0 or 1	Basic scenario (widely used in ATPG)
Bridging Faults	Short circuit between multiple lines	Critical in high-density circuit layouts
Transistor-Level Faults	Transistor failure or degradation	Requires detailed transistor-level analysis
Delay Faults	Violation of timing characteristics (signal propagation delay)	Important for high-frequency pipelined systems
Floating-Gate/Floating Nodes	Unstable inputs, where source-drain resistance may lead to unpredictable values	Common in unstable manufacturing processes

Coupling Faults	One line influencing another (crosstalk, parasitic interference)	Critical for memory and densely packed logic arrays
Supply Faults	Unreliable power supply leading to incomplete logic levels	Critical for low-power systems

The stuck-at fault model is considered the most commonly used and relatively universal due to several reasons:

1. **Simplified formalization:** Any line can be assumed to be either functioning correctly (changing from 0 to 1 and back) or stuck.
2. **High correlation with real defects:** Even in modern manufacturing processes, despite the increasing variety of faults, permanent failures (such as broken interconnects or transistor failures) often conform to the stuck-at model.
3. **Broad tool support:** Automatic test pattern generators (ATPG) and most commercial simulators (e.g., Synopsys, Cadence, Siemens) are primarily designed for detecting stuck-at faults.

However, an important aspect to consider is that not all stuck-at faults result in functional failures. If a fault is located in components such as branch prediction logic or auxiliary power management blocks, its impact may not be observable at the user instruction level [5]. In applied (field) testing, it is crucial to prioritize the evaluation of critical stuck-at faults that lead to incorrect architectural results and potential data integrity issues [4].

The following test coverage metrics are used:

1. **Stuck-At Fault Coverage:** Measures the percentage of all possible stuck-at faults (typically considering both stuck-at-0 and stuck-at-1 at each node) that can be detected by a given set of test vectors or programs. This is calculated as a ratio, and for complex SoCs and processor modules, the number of such faults can reach hundreds of thousands or even millions [1].
2. **Toggle Coverage as a Proxy Metric:** Measures how many different nodes in the circuit have transitioned at least once from 0 to 1 and vice versa during test execution. The rationale is that if a node does not switch states, the presence of a stuck-at fault in that node is unlikely to be detected at the output. Therefore, increasing toggle coverage is expected to correlate with improved stuck-at fault detection.

In the context of field testing, toggle coverage can be efficiently measured in a single gate-level simulation run, whereas a full stuck-at fault coverage evaluation requires separate simulations for each potential fault or computationally intensive algorithms. Using toggle coverage as a proxy metric thus provides significant computational savings, especially during algorithm training [2, 9].

Research Results: Optimization of semiconductor testing using RL-SBST for stuck-at fault models to improve fault coverage

This section focuses on methods for optimizing semiconductor device testing based on the stuck-at fault model and the key proxy metric—toggle coverage. This approach is particularly relevant for complex RISC-V cores, where classical methods such as scan chains and ATPG are either too costly or impractical for field testing conditions [4].

As demonstrated in previous studies [1, 2, 4], a circuit node that has not transitioned between 0 and 1 cannot be detected as being "stuck" in the opposite state. Conversely, if a node undergoes multiple logical transitions, the probability of revealing a fault stuck at a single level increases. A full evaluation of stuck-at fault coverage at each test generation step, particularly in real-time conditions, requires repeated simulations of all potential faults, making it computationally prohibitive [7].

To address this, several modern approaches use toggle coverage as a proxy for stuck-at fault coverage [2]. The main arguments supporting this choice include:

- **Accessibility:** it can be measured in a single gate-level simulation run.
- **Scalability:** even with a large circuit (many gate nodes), measuring toggle transitions in one run is computationally simpler than exhaustive fault enumeration.
- **Correlation with real defects:** sufficient activation of internal nodes, particularly those involved in different execution paths and pipeline stages, increases the likelihood of detecting most functionally critical faults.

Reducing system downtime is a fundamental requirement in field testing. SBST (Software-Based Self-Testing) addresses this challenge by executing test instructions directly on the target processor core without switching the chip into scan mode. The SBST strategy is based on the following principles:

- **Short test programs:** instead of relying on hundreds of thousands of vector-based patterns used in scan testing, SBST solutions typically use fewer instructions while ensuring higher engagement of various functional blocks within the core.
- **Configurability:** test programs can be adapted to specific architectures (in-order/out-of-order execution, pipeline depth, cache presence, etc.).
- **Minimal hardware overhead:** only a small firmware component is required to execute the test and compare the results with a "golden" reference [1].

The main idea behind the RL-SBST (Reinforcement Learning for SBST) approach is to construct a training environment where the reward is based on the increase in the proportion of toggled nodes. At each step, the agent selects the next processor instruction and receives positive reinforcement if the number of newly toggled nodes increases [2].

Several studies utilize the Proximal Policy Optimization (PPO) algorithm due to its scalability in parallel environments [9]. Parallel simulation, such as multiple processor cores or a cluster of computing nodes, enables the generation of a statistically significant set of instructions within a reasonable time frame, bringing the agent’s policy closer to the optimal one while avoiding overfitting to limited scenarios [2].

Since a stuck-at fault cannot manifest unless the corresponding node switches states, an increase in toggle activity indirectly enhances the likelihood of fault detection. Up to a certain threshold, this correlation is strong, as confirmed by practical experiments: programs exhibiting high toggle coverage often show a noticeable improvement in stuck-at coverage.

However, it is important to note that 100% toggle coverage does not equate to 100% stuck-at fault coverage. Some nodes may be inaccessible at the level of user instructions, or certain faults may be masked during program execution [7].

The RL-SBST method can be applied to various RISC-V architectures, including:

- **In-order (InO):** a five-stage pipeline implementation (Fetch, Decode, Execute, Memory, Write-back). This relatively simple microarchitecture maintains a closer relationship between the instruction sequence at the ISA level and internal states.
- **Out-of-order (OoO):** a more complex superscalar core capable of reordering instruction execution to improve performance [4]. This introduces a "gap" between the architectural state and microarchitectural nodes, requiring more refined control when generating test instructions [3].

For gate-level testing, a synthesized netlist of the RISC-V core is used, where each connection is tracked for toggling activity. The length of the test program is typically constrained (e.g., 200 instructions) to enable fast and repeated model simulation [2, 5].

To better illustrate the position of RL-SBST among existing methods, Table 2 summarizes the main approaches to generating test programs for stuck-at faults.

Table 2. The main approaches to creating test programs for stuck-at faults. [1, 2, 7, 8].

Method	Main Optimization Target	Advantages	Disadvantages
Random	Randomly selected instructions	Easy to implement, requires minimal computational resources	Low fault coverage, lacks targeted activation of specific blocks
RISCV-DV	Enhanced randomization of RISC-V instructions	Uses heuristics to generate valid branches and addresses	Limited improvement over pure randomness, may not achieve high stuck-at fault coverage
Bounded Model Checking (BMC)	Formal proof of activating required circuits	High accuracy, theoretically complete coverage (for short tests)	Exponential increase in computational complexity, impractical for long programs and complex OoO architectures

Evolutionary Algorithms	Gradual optimization of the initial test sequence	Relatively simple to implement, allows step-by-step improvement in coverage	Prone to local optima, solution quality heavily depends on the initial state
RL-SBST	Maximization of toggle coverage (proxy for stuck-at faults)	High fault coverage, ability to account for internal circuit states, efficient parallelization	Requires complex gate-level simulations, potential incomplete coverage of inactive blocks if no corresponding instructions exist

The table illustrates that the RL-SBST approach provides a significant advantage, provided that gate-level simulation of the entire core can be performed multiple times. Otherwise, when computational resources and time are limited, compromises must be made, such as reducing program length or lowering model accuracy.

Comparison with existing methods

This section presents the results of a practical evaluation of the proposed RL-SBST approach, comparing it with traditional test program generation methods (Random and RISCV-DV) in testing RISC-V processor cores. This experimental analysis clearly demonstrates the increase in toggle coverage and stuck-at fault detection as the test program length increases, highlighting the advantages of reinforcement learning [2, 3, 7].

Figure 2 illustrates the structure of the RL-SBST platform. It is based on gate-level simulation of the target processor core to capture the exact switching status. The current toggle coverage (TC) and the generated instruction sequence (IS) serve as the input state for the RL model. The RL model interprets changes in toggle coverage as a reward value. Throughout the training process, the deep neural network (DNN) model outputs action data for each step, prescribing a new instruction. Each instruction is then incorporated into the sequence and evaluated using gate-level simulation of the target core. To ensure reproducibility, the test program length is fixed. The total runtime for each model is limited to approximately 200,000 clock cycles, which is sufficient to execute the entire sequence considering the pipeline.

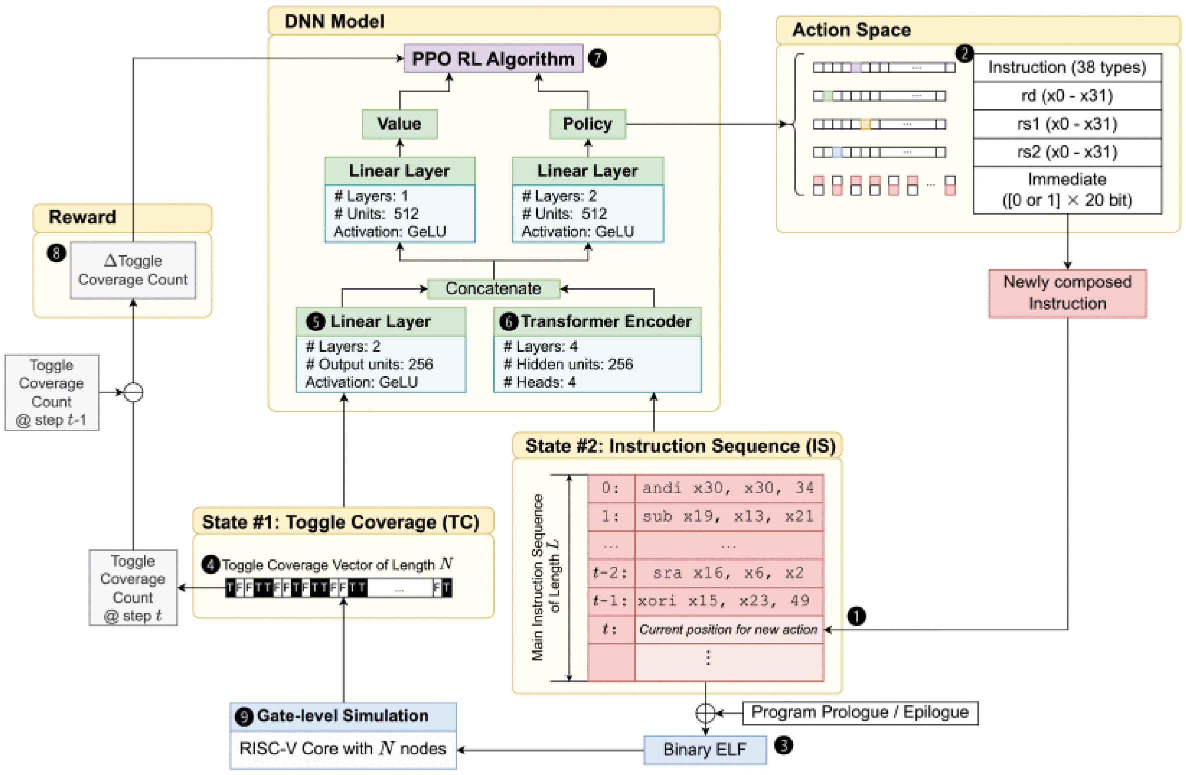

Fig.2. General structure of the RL-SBST environment for generating SBST [1].

As seen in Figure 2, RL-SBST generates a test program by adding one RISC-V instruction per RL step, with each new instruction becoming the next in the sequence. The RL model generates multiple probability distributions for five instruction components: type, three register identifiers, and an immediate value. For example, the generated instruction type is selected from a probability distribution covering 38 potential instruction types in RV32I [1].

For a small number of instructions (L≤50), all methods show relatively modest results (below 50–60% toggle coverage). However, as L increases to 200, RL-SBST surpasses 80%, while random generation rarely exceeds 50% [1,2].

For a complex OoO core, the overall toggle coverage is generally higher due to superscalar execution. However, the gap between methods remains: Random achieves approximately 70% at L=200, RISCV-DV slightly exceeds this value (by about +0.5–1%), while RL-SBST reaches 80–85% [7]. As noted by Dixit H. D. et al. [4], this is a crucial metric for testing given the complexity of the logic.

Table 3 summarizes the aggregated results for stuck-at fault coverage at L=200.

Table 3. Generalized results for stuck-at fault coverage at L=200 [1, 6, 7, 8].

Method	InO Core (approximately 13K nodes)	OoO Core (approximately 58K nodes)	Comment
Random	~75%	~70–73%	Simple implementation, weak optimization
RISCV-DV	~77–78%	~72–74%	Uses heuristics for valid transitions but does not consider microarchitectural information
RL-SBST	~85%	>90%	Adapts to specific nodes, generating instructions that increase their switching
BMC (2-3 cycles)	Can achieve very high local coverage but is not applicable for long tests	Does not scale for long programs or complex cores	Exponential growth in computational complexity

As shown in Table 3, RL-SBST outperforms randomly generated test programs in terms of stuck-at coverage, particularly in complex scenarios. While formal methods (such as BMC) provide high accuracy, they are not suitable for long tests due to state explosion. In general, 80–90% toggle coverage is usually comparable to 70–85% stuck-at coverage [2]. Adding more instructions (beyond 200) can further increase coverage, but it introduces the issue of prolonged test time, especially in parallel RL training.

Future advancements in processor and semiconductor device testing should focus on hybrid strategies that combine reinforcement learning with more detailed fault models. In particular, the integration of stall models (hang or stall phenomena in faults) can help identify the most vulnerable points in the circuit, where traditional stuck-at models do not capture all possible failure scenarios. Incorporating such stall models into RL training would enable the generation of instructions targeting not only classic stuck-at faults but also states where logical elements or pipeline stages "hang" in an incorrect execution mode. This would enhance confidence in the system's reliability under unusual or extreme conditions.

The following recommendations can be proposed:

- **Expansion of fault models:** In addition to stuck-at and stall models, it is necessary to consider transition faults and bridging faults.
- **Hybrid test planning:** Using ATPG can optimize stall fault models, improving fault tolerance in sub-5nm semiconductor designs.
- Implement distributed learning with hardware-in-the-loop support (FPGA-in-the-loop) to accelerate data collection on toggle coverage and speed up RL model training.

- Develop a mechanism for rapid RL-agent configuration for different RISC-V variants or other ISAs, using meta-parameters that generalize and transfer knowledge about the most "problematic" node types and faults between models.

Conclusion

The article examines methodological aspects of testing RISC-V processor cores in field conditions, where it is crucial to balance fault coverage and testing costs. The RL-SBST approach is proposed, combining reinforcement learning and the toggle coverage metric as a proxy for stuck-at fault coverage.

A high level of stuck-at fault coverage (over 90% on the OoO core) has been achieved with a test length of approximately 200 instructions, surpassing random methods and RISCV-DV.

A correlation has been established between the increase in toggle coverage and stuck-at fault coverage, confirming the feasibility of using switching activity to reduce computational costs during RL agent training.

The RL-SBST approach enables the generation of short but effective SBST programs applicable in systems where testing must be performed without complex interventions, such as scan-based methods.

Thus, the proposed method demonstrates the effectiveness of applying machine learning for targeted coverage of potentially problematic nodes while utilizing a proxy metric to reduce costs compared to traditional direct computation of stuck-at fault coverage.

References

[1]. Seo J., Cho H. Generating Efficient Instruction Sequence for Software-Based Self-Testing of Processor Cores using Reinforcement Learning - IEEE Access. – 2024. – pp. 189288 – 189296.
[2]. Chen C. Y., Huang J. L. Reinforcement-learning-based test program generation for software-based self-test, 2019 IEEE 28th Asian Test Symposium (ATS). – IEEE, 2019. – pp. 73-735.
[3]. Zhang Y. et al. Software-based self-testing using bounded model checking for out-of-order superscalar processors, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2019. – Vol. 39 (3). – pp. 714-727.
[4]. Dixit H. D. et al. Silent data corruptions at scale, arXiv preprint arXiv:2102.11245. – 2021. – pp.1-8.
[5]. Hochschild P. H. et al. Cores that don't count, Proceedings of the Workshop on Hot Topics in Operating Systems. – 2021. – pp. 9-16.
[6]. Deligiannis N. I. et al. Automating the generation of programs maximizing the repeatable constant switching activity in microprocessor units via MaxSAT, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2023. – Vol. 42 (11). – pp. 4270-4281.
[7]. Zhang Y. et al. BMC-Based Temperature-Aware SBST for Worst-Case Delay Fault Testing Under High Temperature, IEEE Transactions on Very Large Scale Integration (VLSI) Systems. – 2022. – Vol. 30 (11). – pp. 1677-1690.
[8]. Deligiannis N. I., Cantoro R., Reorda M. S. Maximizing the switching activity of different modules within a processor core via evolutionary techniques, 2021 24th Euromicro Conference on Digital System Design (DSD). – IEEE, 2021. – pp. 535-540.
[9]. Pfeifer N. et al. A reinforcement learning approach to directed test generation for shared memory verification, 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE). – IEEE, 2020. – pp. 538-543.
[10]. Kochte M. A., Wunderlich H. J. Self-test and diagnosis for self-aware systems, IEEE Design & Test. – 2017. – Vol. 35 (5). – pp. 7-18.