Directed Test Generation for Hardware Validation: A Survey

ARUNA JAYASENA, University of Florida, USA
PRABHAT MISHRA, University of Florida, USA

The complexity of hardware designs has increased over the years due to the rapid advancement of technology coupled with the need to support diverse and complex features. The increasing design complexity directly translates to difficulty in verifying functional behaviors as well as non-functional requirements. Simulation is the most widely used form of validation using both random and constrained-random test patterns. The random nature of test sequences can cover a vast majority of scenarios, however, it can introduce unacceptable overhead to cover all possible functional and non-functional scenarios. Directed tests are promising to cover the remaining corner cases and hard-to-detect scenarios. Manual development of directed tests can be time consuming and error-prone. A promising avenue is to perform automated generation of directed tests. In this paper, we provide a comprehensive survey of directed test generation techniques for hardware validation. Specifically, we first introduce the complexity of hardware verification to highlight the need for directed test generation. Next, we describe directed test generation using various automated techniques, including formal methods, concolic testing, and machine learning. Finally, we discuss how to effectively utilize the generated test patterns in different validation scenarios, including pre-silicon functional validation, post-silicon debug, as well as validation of non-functional requirements.

CCS Concepts: • **Hardware → Functional verification**; *Bug detection, localization and diagnosis*; *Test-pattern generation and fault simulation*;

Additional Key Words and Phrases: hardware verification, test generation, functional validation, security validation

1 INTRODUCTION

Computing devices rely on both hardware and software to provide the required functionality. The complexity of the hardware is increasing rapidly due to technological advances as well as the demand for supporting complex and heterogeneous features in diverse applications, ranging from simple handheld devices to complex autonomous systems. These systems utilize System-on-Chip (SoC) to provide the computing backbone. A typical SoC design today includes hundreds of heterogeneous components consisting of billions of transistors. In order to design such complex hardware in a tight time-to-market window, there are new specification languages, novel design and synthesis flows, and efficient tools for physical design and fabrication. A major bottleneck in hardware design methodology is how to verify numerous functional behaviors and complex non-functional requirements. Figure 1 shows validation steps during SoC design methodology.

Simulation is the most widely used form of validation using random as well as constrained-random tests. While random tests are suitable for covering a vast majority of easy-to-detect scenarios, they cannot cover complex corner cases and hard-to-detect scenarios in a reasonable time. Directed tests are promising to cover the remaining scenarios. Manual development of directed tests can be time-consuming and error-prone. Recent efforts address this challenge by focusing on automated test generation techniques. This paper provides a comprehensive survey of existing approaches for automated generation of directed tests for hardware validation.

The remainder of this section is organized as follows. Section 1.1 provides an overview of hardware design methodology. Section 1.2 highlights the importance of hardware validation followed by an overview of existing validation methods in Section 1.3. Section 1.4 compares this paper with related surveys. Finally, Section 1.5 presents the survey methodology.

Authors' addresses: Aruna Jayasena, University of Florida, Gainesville, FL, 32611, USA, arunajayasena@ufl.edu; Prabhat Mishra, University of Florida, Gainesville, FL, 32611, USA, prabhat@ufl.edu.

1.1 Overview of Hardware Design Flow

Hardware designs go through several stages during the design cycle, starting from specification to a physical chip (integrated circuit). Figure 1 illustrates the major stages involved in the hardware design flow. The specification stage captures the functional intent as well as design constraints using a specification language. Usually, the specification is captured using Transaction Level Modeling (TLM), such as SystemC TLM models [43]. The specification needs to be validated to ensure that it can be used as a golden reference model. The next stage involves the Register-Transfer Level (RTL) implementation of the specification using Hardware description languages (HDL), such as VHDL [5] or Verilog [121]. The implementation needs to capture both the structure and behavior of the target design [119]. The RTL implementation needs to be validated to ensure that the implementation satisfies the specification. The next stage involves the synthesis of the RTL implementation to produce a gate-level implementation. Hardware synthesizers are used for the synthesis process and the output should be verified again with the RTL implementation as well as the specification to ensure that the synthesis process did not introduce any bugs or potential vulnerabilities. The validated gate-level implementation goes through physical design tools, such as layout and placement. The layout is fabricated to produce the integrated circuit (IC). Finally, the IC goes through extensive testing and post-silicon debug to check for bugs that escaped pre-silicon validation as well as manufacturing defects. In this survey, we focus on test generation methods for pre-silicon as well as post-silicon validation. Note that test generation for detecting manufacturing defects (manufacturing testing) is beyond the scope of this survey.

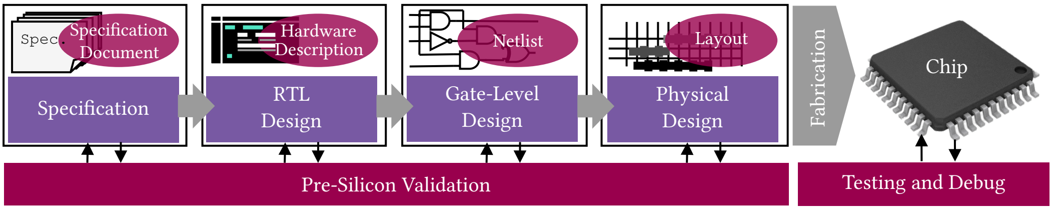


Fig. 1. Hardware Design Flow. Validation is important across the design cycle.

It is important to note that the design is validated in each stage during the design flow since each transformation creates the potential to introduce functional bugs and unexpected scenarios. We also need to check for non-functional requirements, such as area, power, performance, real-time constraints, and security vulnerabilities. The test/validation engineers create test cases to check whether the actual output (e.g., simulation of RTL implementation) matches with the expected behavior (e.g., simulation of the golden specification). Simulation cannot guarantee the absence of bugs due to the complexity of checking all possible (exponential) scenarios. For example, let us consider a simple 64-bit adder circuit. The inputs to the adder are two 64-bit numbers and the output is a 64-bit number. In order to verify all possible scenarios, we need to have 2^{128} test cases, which is infeasible. In order to validate designs with acceptable guarantees without exhaustive testing, efficient test generation and test reuse techniques are required.

1.2 Hardware Validation Importance and Trends

According to the functional verification study conducted by Wilson Research Group, design testing and verification effort is a significant contributor to overall cost [38]. Figure 2 shows the average number of engineers utilized by the semiconductor companies to design and verify the product. It is clear that the ratio between verification engineers to design engineering is increasing over the years. This signifies that the verification effort required by the companies for ASIC designs is increasing at a faster rate than the increase in design complexity.

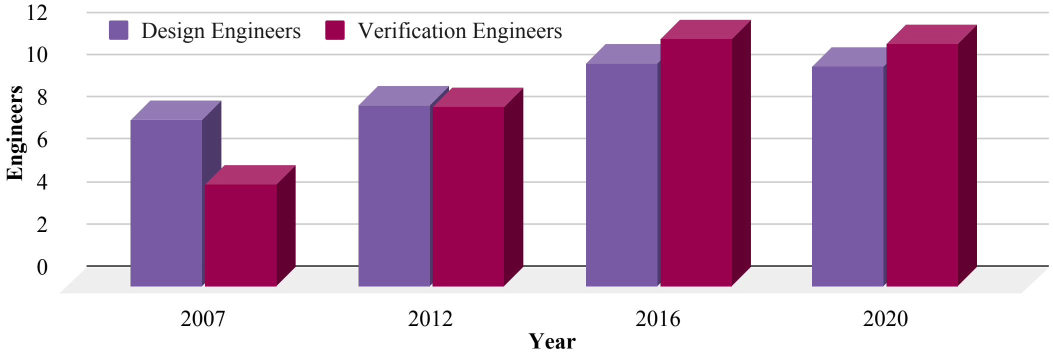


Fig. 2. Average number of design engineers and verification engineers utilized by ASIC projects [38]

In spite of these verification efforts, still bugs escape from the verification stage causing design re-spins that introduce considerable financial impact to the semiconductor companies. Figure 3 presents different types of undetected bugs in the fabricated ICs that led to the silicon re-spin (bug fix or re-design and expensive re-fabrication). While different flaws can lead to re-spin, functional bugs are the lead contributor.

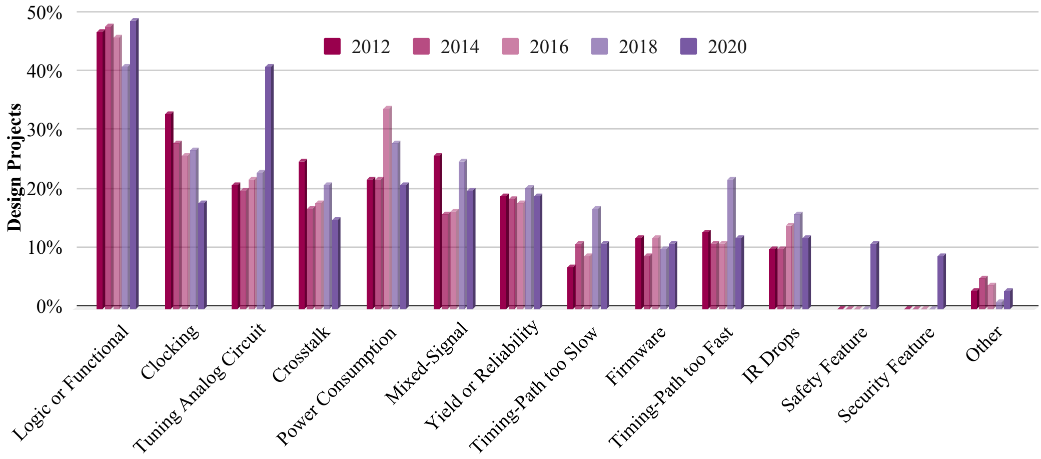


Fig. 3. Different categories of design flaws that led to bug escapes and silicon re-spin [38]

The research study highlights the types of escaped bugs in spite of extensive pre-silicon validation. It also signifies the hardware validation complexity and highlights the need for fast, scalable, and automated hardware validation techniques.

1.3 Overview of Hardware Validation Methods

Simulation-based validation is the commonly used approach by semiconductor companies. Simulation with random test cases can identify most of the easy-to-detect bugs quickly [12, 33, 59, 133]. However, the remaining hard-to-activate scenarios and corner cases [21] contribute to the root cause of the bug escapes [38], as shown in Figure 3. In order to validate the remaining hard-to-activate corner cases, there are two choices: (i) keep running random tests and hope it will detect the remaining bugs, or (ii) perform coverage analysis (e.g., branch coverage) and construct custom (directed) tests to activate the uncovered scenarios.

The probability of covering the remaining hard-to-activate scenarios is very low using random test cases even if the designers are able to invest significant validation effort (time) [21]. Note that there are timing constraints for each phase in the design cycle that prohibit unlimited verification cycles. Therefore, a practical solution is to construct directed tests to cover the remaining hard-to-activate scenarios. Unfortunately, manual construction of the directed tests can take a considerable amount of time and effort. Most importantly, it may not be feasible to write a directed test for large designs and complex corner cases. A promising alternative is to perform automated generation of directed tests. This paper surveys these automated test generation techniques.

There are different types of directed test generation techniques proposed over the years for hardware validation. The effectiveness of all these techniques can be measured with several factors.

(1) Design Coverage: The percentage of the functional or non-functional requirements covered by the constructed test cases. For example, a design has 10,000 branches, and the random tests covered 80% of them. We need to generate 2000 directed tests (assuming no test compaction is possible) to cover the remaining 20% branches.
(2) Scalability: It measures the scalability of the test generation technique for verifying large industrial designs. For example, a test generation method based on model checking may perform well for designs with a few thousand gates but may lead to state space explosion when dealing with million gate designs.
(3) Overall Effort: This includes both test generation effort and validation effort. The test generation time is negligible for random tests, but we need to consider the time for millions of simulations (assuming millions of random tests). On the other hand, directed test generation takes a long time compared to random test generation, however, the total number of simulations can be in the order of a few thousand (assuming a few thousand directed tests can provide the same coverage).

In practice, the designers use an effective combination of simulation-based validation and formal verification. Specifically, they try to verify small and critical components through formal methods, while using simulation-based validation using random or constrained-random tests to verify larger modules as well as the overall design. While code coverage is a good metric for software verification due to its sequential nature, code coverage is not enough for hardware verification since it involves concurrent execution of multiple modules. In other words, even perfect (100%) line coverage provides limited insight into the effectiveness of testing concurrent finite state machines. Automated techniques surveyed in this paper offer promising benefits, such as faster coverage of targeted scenarios and enhanced test generation efficiency, contributing to an effective and comprehensive hardware verification.

1.4 Major Differences with Existing Surveys

There are several surveys related to hardware test generation and validation methods. The most recent work [131] discusses hardware verification with assertions and test generation methods for activating assertions. There are surveys for hardware verification using formal methods [25, 45, 46]. Coverage-based test generation techniques are surveyed in [54]. There are also hybrid techniques that target state explosion problem for functional verification [10]. The existing surveys have several limitations: (i) they are very specific (e.g., test generation for assertion coverage [131] or machine learning based test generation [54]), (ii) they do not cover test generation techniques (e.g., formal verification surveys [45]), and (iii) they do not cover recent approaches (e.g., published more than 10 years ago [10, 54]). *To the best of our knowledge, there are no recent comprehensive surveys on directed test generation for hardware validation.*

1.5 Survey Methodology

Figure 4 outlines our directed test generation based hardware validation survey methodology. It consists of three major parts: test generation techniques, test translation methods, and use cases. Section 2 outlines different test generation techniques proposed in the literature. Specifically, we survey directed test generation techniques that utilize formal methods, concolic testing, machine learning, constrained random methods, statistical methods, and automated test pattern generation (ATPG). Section 3 surveys hardware test translation between different abstraction levels. Section 4 surveys use cases of directed tests in different validation scenarios, including pre-silicon validation, post-silicon validation, security validation, and validation of non-functional requirements. Finally, we conclude the paper in Section 5.

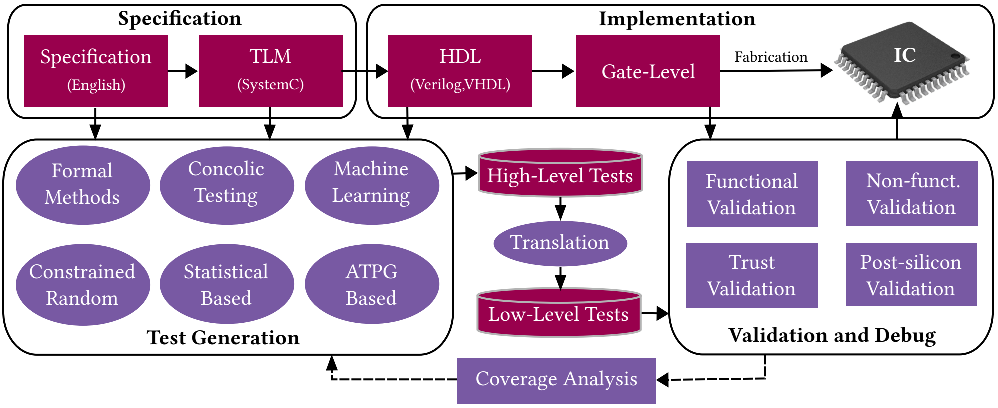


Fig. 4. Overview of directed test generation: Tests are generated based on the specification. These high-level tests are then translated to low-level tests in order to apply them directly to the implementation.

2 DIRECTED TEST GENERATION METHODS

Directed test generation techniques provide an automated framework for generating efficient tests. These techniques consist of algorithms that take design information and constraints into account in order to generate efficient and directed test patterns. The existing directed test generation techniques can be broadly divided into the following six categories based on the core method used for test generation: formal methods, concolic testing, machine learning, constrained-random, statistical, and ATPG. The remainder of this section surveys test generation techniques for each of these categories. Figure 24 will compare these categories based on coverage, scalability, and test generation effort.

2.1 Test Generation using Formal Methods

Formal verification is a complementary approach to simulation-based validation. Formal methods, such as model checking [19, 27, 63, 89], theorem proving [69], and equivalence checking [11], are widely used to provide verification guarantees [60]. In this section, we survey approaches that utilize model checking for automated generation of directed tests. Model checking based test generation relies on counterexample generation. For example, if we want to generate a test to stall the decode unit of a processor, we need to write the expected behavior in the form of a temporal logic property. The model checker will take the negated version of the property (decode cannot be stalled) and the design as inputs, and will produce a counterexample. The generated counterexample can be used as the test to stall the decode unit. While the basic idea is simple, model checking can lead to state space explosion in the presence of large designs and complex properties. To address this challenge, the researchers have explored satisfiability (SAT)-based bounded model checking for test generation.

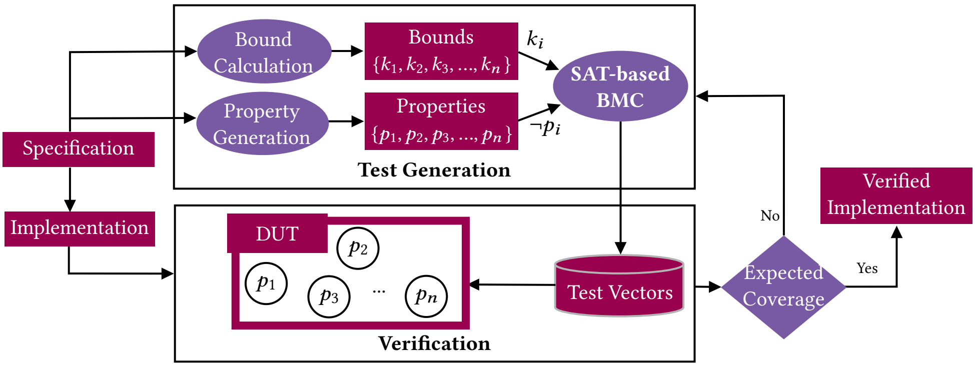


Fig. 5. Overview of directed test generation using bounded model checking

Bounded Model Checking (BMC) tries to find a counterexample for a given property within a given bound [24]. Let us assume that we were given a design D and a property p with a bound of k. BMC will use Equation 1 with D unrolled for k cycles to encode the design for satisfiability solving.

$$BMC(D, p, k) = I(s_0) \wedge \bigwedge_{i=0}^{k-1} R(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{k} \neg p(s_i) \tag{1}$$

Here $I(s0)$ denotes the initial state of the design, state transition from s_i to s_{i+1} is represented by $R(s_i, s_{i+1})$ while $p(si)$ monitors the property p status during state s_i. Equation 1 is converted into Conjunctive Normal Form (CNF) and uses SAT solvers to find a suitable assignment. If CNF finds an assignment that means the p does not hold within the given bound with k cycles otherwise, it can be concluded that p holds up to k ($D \vDash_k p$). There are multiple BMC-based directed test generation techniques proposed in literature [18, 22, 64, 88, 107, 113, 114, 120]. An overview of SAT-based BMC is illustrated in Figure 5.

Test Generation using Satisfiability (TEGUS) was proposed by Stephan et al. in [120]. TEGUS sorts the design variables using depth-first search according to the depth they appear in the design. This simplifies the selection of variables for the next unsatisfied clause at each branch statement in the design.A common problem that encountered when using SAT-based bounded model checking is determining the correct bound for each property. Overestimation of the bound can lead to an exponential number of possibilities. Koo and Mishra proposed a technique to improve the process of calculating the bounds for properties [64]. The authors have shown that the upper limits of the bounds are the longest computation paths in the pipeline. Even the longest computation path was an overestimation of the bound for other possible paths; therefore, the authors tightened the bound by calculating the temporal distance between the root node and the node under verification. To reduce the test generation complexity, Koo et al. [63] introduced a design and property decomposition framework for pipelined processors. The test vectors were generated using SAT-based BMC. Figure 6 presents how the properties were decomposed in the proposed technique. The decomposed properties were fed into the BMC by inverting the property ($\neg P$) with an estimated bound (k) corresponding to the property. The authors explored two techniques for property decomposition: (i) path-level (vertical) partitioning, and (ii) stage-level (horizontal) partitioning of pipelined processors. The tests were generated based on the above partitions, and the generated tests were composed to produce the final test for the original property.

Mishra and Chen proposed an alternate solution for reducing Model checking complexity during test generation [88]. They utilized incremental satisfiability for directed test generation using

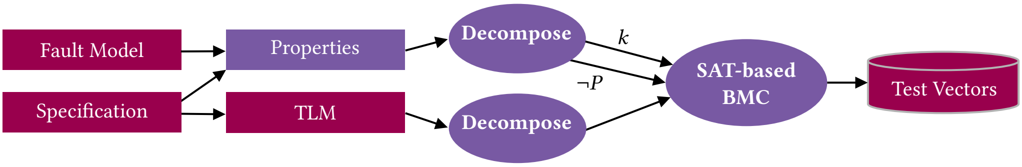


Fig. 6. An overview of property decomposition technique in [64] and [63]

multiple properties. The proposed method consisted of efficient techniques for property clustering as well as selective forwarding of conflict clauses. In property clustering, the authors analyzed the properties based on the design structure and the fault model. For example, properties were categorized into one group if they aimed to cover a specific area of the design with the same fault model. Next, a base property was selected, and potential conflict clauses that could be used in other properties were identified. The results showed that this approach significantly reduced the test generation time compared to *zChaff* [92] for the MIPS processor benchmark.

Chen et al. [22] tried to address some of the major bottlenecks in SAT search, including constraint propagation and long-distance backtracking. They proposed a decision-ordering technique that involved clusters of similar properties. The proposed decision-ordering heuristic exploited the assignment of previously generated tests and incorporated it into the next test generation effort, reducing the time and effort taken to generate the test vectors. Specifically, the authors tried to reduce the number of implications and conflict clauses of unchecked properties using the learned knowledge from previously checked properties. The authors explored two heuristics for decision ordering: bit value ordering and variable ordering. The proposed test generation algorithm took the formal model of the design and the cluster of similar properties as inputs. Next, it generated the CNF of the base property and solved the base property. Then it constructed the CNF for the unchecked properties and updated the satisfiable assignments from previous assignments. The authors evaluated the proposed method on a MIPS processor that generated more than five times improvement over *zChaff* [92] and [88].

Qin et al. proposed an improved test generation methodology using SAT-based BMC [107]. Satisfiability solvers can take advantage of previously completed calculations for future calculations and also they can solve multiple properties when the bound is known. The authors proposed combining both of the above advantages to reduce the test generation time. The proposed method tried to identify the similarities between different SAT instances for multiple properties and came up with a bound on the same design. Moreover, the knowledge obtained from earlier queries was transferred to the later queries. The authors proposed a synchronized test generation algorithm for multiple properties and incremental SAT-solving techniques to reduce the test generation time. The authors conducted experiments to evaluate the proposed technique, and the results illustrated that the proposed technique achieved on average a four times speedup with respect to related methods.

Chen and Mishra proposed an efficient automated approach to scale down the falsification complexity using property decomposition and learning techniques to address the state explosion problem in SAT-based BMC [18]. They utilized both learning-oriented property decomposition and decision ordering-based learning techniques. Figure 7 illustrates the major differences between test-oriented property decomposition and their learning-oriented property decomposition. Test-oriented property decomposition generates test cases for each property separately and combines them to generate tests while learning-based techniques can combine the previously calculated knowledge for future properties and generate efficient test vectors. The authors decomposed the properties considering both spatial and temporal characteristics. The authors also utilized decision-ordering techniques to optimize the learning process. They showed that the counterexample for a property could be used as a suitable variable assignment to the inputs of the design. The authors used similar kinds of properties for learning as a bias for the decision ordering. Due to the fact that the property decomposition was done based on the capability of utilizing previously learned properties to the later properties, the authors were able to demonstrate a drastic reduction in test generation time and effort compared to the previously discussed techniques.

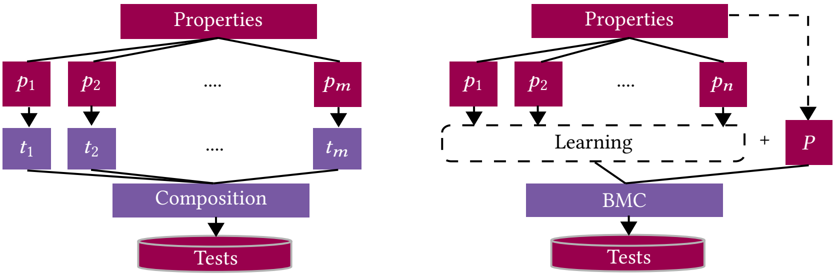


Fig. 7. Comparison between test-oriented test generation (Left) and learning-oriented test generation (Right)

While the approaches discussed above uses formal methods for test generation, HIVE [57] utilizes test cases to provide formal guarantees. Jayasena and Mishra proposed a hint-based symbolic verification framework that accepted a test plan and provided formal guarantees for each test case using symbolic evaluation [57]. The authors demonstrated the effectiveness of the proposed technique on various RISC-V-based hardware implementations and were able to find functional issues with various peripheral devices.

2.2 Test Generation using Concolic Testing

Although test generation using formal methods provides a guarantee to activate the expected scenario, it has an inherent limitation of state explosion problem. When the design and property are complex, test generation techniques based on formal methods may fail due to the capacity limitations of formal methods. On the other hand, simulation with random test is fast and scalable, but it does not provide any guarantee of activating a specific scenario. Concolic testing combines the benefits of both worlds. Specifically, it effectively combines concrete simulation and symbolic execution.

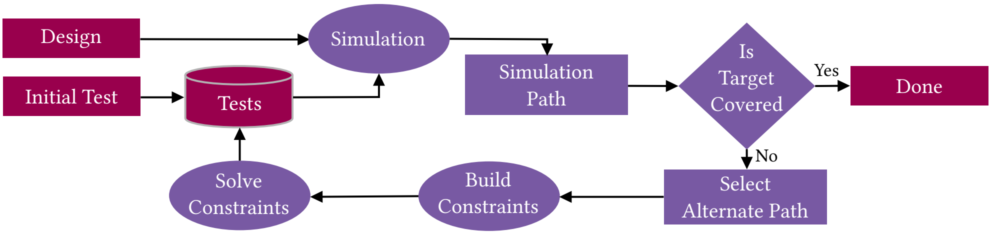


Fig. 8. An overview of concolic testing based test generation.

Figure 8 illustrates the major steps involved in concolic testing. It starts with an initial set of test vectors (e.g., random tests). The instrumented design is simulated with the initial test and the simulation path is recorded. If this covers the target (expected scenario) in the design, we have the test pattern. Otherwise, another path should be selected such that it will increase the likelihood of reaching the target quickly. Concolic testing uses symbolic execution to solve path constraints. For this purpose, the Control Flow Graph (CFG) is extracted from the design, and the execution path is selected based on the CFG. Efficient heuristic techniques for path selection were proposed in the literature [2, 42, 74, 74–76, 79, 110]. All these techniques considered basic blocks for path selection during simulation. A basic block is a block of codes that does not have any internal branches. Figure 9 compares different concolic testing approaches. Figure 9a shows an instance where simulation was performed with random test vectors. This will cause to take different simulation paths with different test vectors and may not cover hard-to-activate targets.

Geist et al. presented a hybrid verification methodology that integrated the guarantee of formal verification with quick coverage achieved by verification using simulation [42]. To determine the path traversal, the authors utilized a Binary Decision Diagram (BDD) constructed from the design combined with symbolic execution. For the purpose of directing the test generation procedure toward a specific area, the authors proposed an abstraction mechanism. The abstraction was achieved by partitioning the state variables based on the coverage model set, ignore set, and care set. For processing care set variables, the authors proposed a heuristic based on the register logic transitions. Then, the abstract tests were generated, and a test translation process was employed to obtain the actual test that could be used for simulating RTL models.

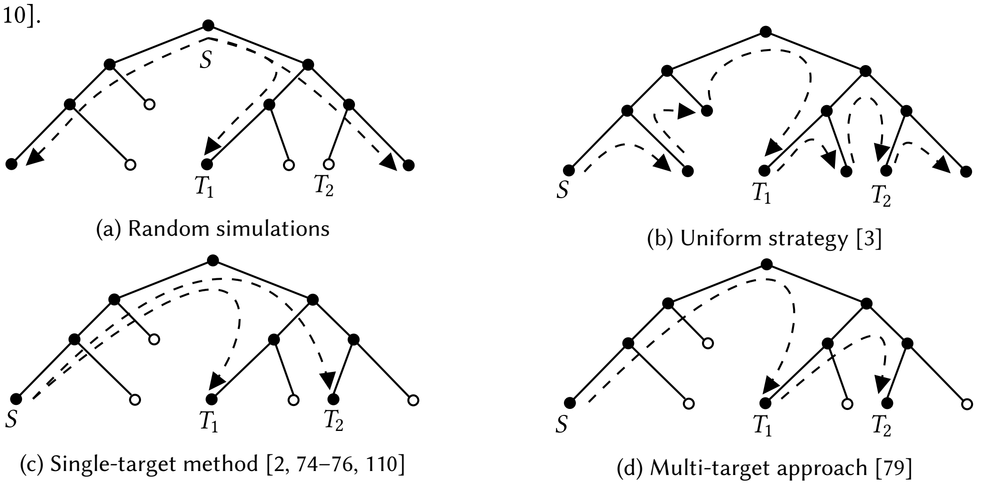
STAR was another concolic testing approach proposed in [74]. The STAR framework used the following steps. First, the designs were instrumented so that they could print the executed statements during simulations. Next, the designs were simulated using initial (random) test patterns. The authors performed symbolic simulations of the RTL statements that were executed during the concrete simulations. Then, the authors performed sequential unrolling of the designs and performed simulations on unrolled designs. During the simulation processes, path exploration and constraint solving were employed to get better statement coverage. Finally, utilizing a constraint stack, mutated or inverted constraints were solved using SMT solvers to obtain input test patterns corresponding to the expected targets of the designs.

HYBRO [75] tried to address the drawbacks of STAR [75]. Although STAR utilized a hybrid test generation mechanism, it still suffered from path space explosion when dealing with large designs. HYBRO utilized branch coverage as a heuristic to address the exponential number of possible paths. The proposed methodology recorded the branch coverage in the CFG as a heuristic to stimulate all reachable branches in CFG. Although this did not guarantee 100% coverage, HYBRO outperformed STAR with a reduction in test generation time. In the instances where STAR failed due to the path explosion problem, HYBRO was able to generate test vectors due to the coverage heuristic.

Liu and Vasudevan [76] proposed a concolic testing approach with a method to cache the symbolic state data to mitigate the path explosion problem in STAR[74] and HYBRO [75]. In the proposed method, bitmap encoding was used to store the data about explored symbolic states. This facilitated ease of comparison during subsequent simulations. A sub-path was subjected to pruning when an explored symbolic state was reached again in later symbolic execution. Further, the authors proposed two optimization techniques: dynamic Use-Def chain slicing and local conflict resolution, to improve the efficiency of test generation. Use-Def chains could connect a use of a variable with all possible definitions of the variable throughout the design. The proposed improvements significantly reduced the test generation time compared to [74, 75].

There were also other approaches to improve the performance of [75]. Qin and Mishra [110] proposed a concolic testing framework by combining static analysis and simulation-based validation. Unlike [75], this approach was able to handle dynamic array references, which is a common failure point of model checkers due to the state space explosion problem. This approach instrumented the design-under-test so that the tool could keep track of the simulation traces. Next, the instrumented design was simulated. By analyzing the traces, new constraints were generated so that the algorithm could compute the coverage. The tests were generated by solving the constraints. These steps were repeated until the expected coverage was achieved. The results demonstrated that the proposed test generation method could outperform [75] in terms of branch coverage.

A major challenge in concolic testing is how to select a profitable branch (path) for the next iteration. In order to avoid the selection of non-beneficial branches, Ahmed and Mishra[3] proposed a Qualifying Event-based Search (QUEBS) heuristic for concolic testing. Figure 9b illustrates the uniform strategy during the search procedure utilized in the proposed technique. This method eliminated the possibility of selecting the same branch repeatedly during a traversal. QUEBS used a counter to monitor each branch with a limit. If the counter was less than the limit, then test patterns were generated using a constraint solver. If an expected target was reached, all the branch counters were set to 0 except for the last selected one. The authors were able to achieve higher coverage within a limited time by preventing repeated selection of the same branch compared with [110].



(a) Random simulations

(b) Uniform strategy [3]

(c) Single-target method [2, 74–76, 110]

(d) Multi-target approach [79]

Fig. 9. Comparison of different path selection approaches in concolic testing. Here S represents the starting branch with the initial test vector and T_1 and T_2 represent two target branches in the design.

Coverage of corner cases is a challenging task in hardware verification. Ahmed et al. [2] proposed a methodology to activate hard-to-activate corner cases in hardware designs using concolic testing. They first constructed the CFG and used this to traverse the design to find the corner case (target) scenarios. Due to the concurrent nature of hardware designs, each process will have different CFGs. To connect CFGs together, the authors introduced an edge realignment step using inter-process dependencies. Next, they employed distance heuristics to denote how close each block was to the target. Starting with random test vectors, the proposed methodology gradually forced the simulations toward activating the target scenario based on the distance heuristics. It could generate test vectors faster compared with the EBMC [93] model checker and also consumed less memory for activating all of the branches where EBMC failed due to state space explosion.

The techniques discussed so far activate one target at a time with concolic testing, as illustrated in Figure 9c. Lyu et al. proposed a technique to activate multiple targets using Concolic testing [79]. Figure 9d illustrates an example of a proposed multi-target approach where the wasted effort is minimized by transferring the learning of the current target to the succeeding target. In order to accelerate the process of test generation in multi-target scenarios, the authors pruned the redundant targets using the CFG. Similar to [2], they executed edge realignment. Finally, they employed target clustering to achieve the best initial paths for concolic testing. Two targets were clustered into the same group if they shared a common simulation path. This clustering technique eliminated overlapping searches in single-target iteration, streamlining the process. The authors evaluated the improvement of the proposed method against [3] and [2], achieving a time reduction of more than 13 times on average.

An end-to-end concolic testing framework for hardware/software co-validation of SoCs was proposed in [17], capturing holistic system-level traces and facilitating custom validation with an instrumentation interface. The authors demonstrated the effectiveness of the proposed framework on prototype implementations on the QEMU emulator by successfully uncovering various bugs. Alam et al. presented the FirVer framework for firmware binary testing [4]. Instead of relying on the source code, the proposed technique performed the test generation on the compiled binary. The authors utilized virtual machines to incorporate target hardware interfaces into the test generation. The proposed technique demonstrated over 90% line coverage on the tested libraries.

2.3 Test Generation using Statistical Methods

Statistical methods for directed test generation are commonly used in hardware verification. When the design under test has an exponential number of scenarios to consider, statistical methods can help to reduce the sample set. For example, statistical test generation techniques are extensively used for hardware Trojan detection and manufacturing testing. The common expectation in both hardware Trojan detection and manufacturing testing is to activate multiple sets of triggers or faults at once. There have been promising statistical test generation schemes developed to activate triggers/faults in hardware designs [16, 52, 80, 105, 118].

N-detection is a statistical test generation technique that maximizes the chances of activating unmodeled faults in manufactured chips. The expectation of the N-detect test is that activating each stuck-at-fault N times will activate all possible unmodeled faults around the considered fault when N is sufficiently large enough. Pomeranz and Reddy [105] proposed a technique to measure the quality of N-detect test sets. Following this, Chakraborty et al. [16] proposed MERO, a statistical method based on the N-detect principle, to activate malicious hardware Trojan triggers. Hardware Trojans can be constructed with any number of combinations of signals.The authors proposed a procedure to statistically activate each single trigger signal N times, suggesting that doing so would likely activate all possible Trojan trigger combinations when N is sufficiently large. They outlined a specific process to obtain test vectors that would activate rare signals N times, aiming to enhance the chances of detecting and activating hardware Trojans. First, the design underwent simulation using random test vectors, during which the number of rare signals activated by each test vector was observed. Following this, the test vectors were sorted in descending order based on their ability to activate rare signals. The algorithm then selected test vectors one at a time from the sorted list, flipping one bit at a time and observing the subsequent activation of rare signals. This methodology showed promising results when applied to ISCAS-85 benchmarks, demonstrating its effectiveness in activating rare signals.

Huang et al. improved upon the ideas presented in MERO, developing MERS for side-channel based Trojan detection, as presented in [52]. Their goal was to maximize the switching activities of rarely triggered signals in circuits, addressing two main objectives crucial for side-channel aware test generation: (i) activating the Trojan circuit, and (ii) minimizing the switching in the rest of the design. Initially, MERS generated test vectors following the steps outlined in [16], but instead of flipping bits, they mutated the test vectors. These generated test vectors were then optimized using hamming distance-based reordering (MERS-h) as well as simulation-based reordering (MERS-s) to enhance side-channel sensitivity. The optimized test vectors outperformed both random (by 96%) and MERO (by 38%) in terms of Trojan coverage on the ISCAS benchmarks.

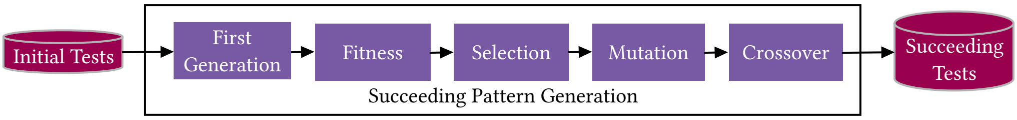


Fig. 10. Genetic algorithms based succeeding pattern generation for maximizing switching sensitivity for side-channel based Trojan detection in [80, 118]

To further enhance the side-channel sensitivity, Lyu et al. utilized genetic algorithms [80]. They constructed pairs of tests, where they generated the first pattern similarly to MERO [16], and then generated the succeeding patterns using a genetic algorithm. Figure 10 presents the steps that were used for generating succeeding patterns to maximize the side channel sensitivity. For searching the succeeding patterns, they utilized initialization, fitness computation, selection, crossover, and mutation with the intention of finding the maximum current switching among the rare nodes. The generated test vectors achieved better coverage than MERS [52] in detecting Trojans.

Recently, Shi et al. presented a side-channel-based test generation technique to detect maliciously implanted Trojans in hardware designs [118]. The authors first identified the potential hardware Trojan trigger nodes and payload nodes. They generated tests to activate the identified potential trigger nodes with relevant inputs to the design identified by correlation analysis. This analysis only considered the static connection between the possible trigger conditions and the design inputs and generated test vectors to manipulate the inputs to activate the Trojan trigger. Next, they utilized test reordering such that the test vectors maximized the Trojan circuit activities while minimizing the design activities. This improved the side-channel sensitivity of the Trojan footprint. The proposed test generation technique was applicable on gate-level designs and the authors related the activities at gate-level with a manufactured chip. Compared to MERO and MERS, it could generate 28.8% more compact test vectors with an improved trigger coverage of 55.4%.

2.4 Test Generation using Machine Learning

Directed test generation techniques rely on heuristics to generate efficient input test vectors. There have been promising efforts in literature to enhance the heuristics with various machine learning techniques. The use of machine learning techniques for directed test generation has been extensively surveyed by [54].

Ravotto et al. introduced a directed test generation method that leverages evolutionary strategies for peripheral cores, focusing on the dynamic extraction of Finite State Machine (FSM) characteristics [112]. The process involved extracting details related to FSM states and transitions dynamically, as the simulation unfolded. The primary objective of this approach was to ensure comprehensive coverage of all conceivable FSM states and transitions. Figure 11 provides an overview of the proposed approach in [112]. The authors developed an evaluator that gathered output data directly from simulations and conducted real-time exploration of the Finite State Machine (FSM). This allowed for an in-depth analysis of how effectively the test vectors, produced by their proposed algorithm, were covering the design. Furthermore, the evaluator played a crucial role in dynamically adjusting the parameters within the evolutionary algorithm to optimize performance. The application of this method yielded impressive results, achieving a fault coverage rate of 91.16% on various benchmarks, including PIA, VDU, and UART.

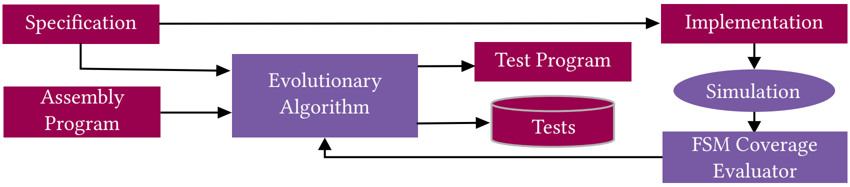


Fig. 11. Directed test generation framework in [112] using dynamic FSM extraction.

The performance and efficacy of MERO [16] are highly reliant on the bit-flipping heuristic it employs to activate a particular rare signal N times. In an attempt to enhance this bit-flipping algorithm, Pan and Mishra incorporated reinforcement learning techniques in their approach [101]. They utilized the Sandia Controllability/Observability Analysis Program (SCOAP) parameters associated with each rare signal, facilitating the determination of the state of the test generating agent within the circuit's environment. The tests produced through this advanced method demonstrated a remarkable improvement in trigger coverage, showing a 77.13% increase compared to the results achieved with MERO. The concept of reinforcement learning, as applied in [101], was also utilized for test generation specifically tailored for delay-based side channel [102]. In this approach, critical path analysis was employed to generate test vectors aimed at maximizing side-channel sensitivity. Authors have used the current test vector as the agent, while the existing circuit is treated as both the design and the action. The approach involves mutating the current test vector through bit flipping on the vector itself. This method yielded significant performance improvements. When compared to random test vectors, the proposed approach demonstrated a 15-fold increase in effectiveness. Moreover, compared against the method outlined in [81], the proposed approach showcased an improvement by a factor of 1.9, illustrating its efficacy in enhancing side-channel sensitivity through reinforcement learning-based test generation.

Vasudevan et al. introduced Design2Vec [123], an architecture for learning continuous representations that captured the semantics of a hardware design at the RTL level. Utilizing the control data flow graph (CDFG) of the hardware combined with graph neural networks (GNN), Design2Vec incorporated a specialized propagation layer to effectively handle the concurrent and non-terminating semantics of RTL. Design2Vec demonstrated its utility in two hardware verification tasks: coverage prediction and test generation, offering faster insights into coverage estimations and efficiently generating tests for hard-to-cover points, thereby significantly reducing simulation time and resource costs. Liang et al. proposed an efficient RTL level test selection methodology using unsupervised learning [72]. Since compiled simulation with software models is much faster, the authors utilized coverage data from functional simulators in software languages to facilitate RTL functional coverage closure. The authors demonstrated improved efficiency and performance, particularly through the application of isolation forest anomaly detection in hardware verification. The methodology showcased its efficacy by significantly reducing RTL simulation runtime and achieving higher test efficiency in hitting critical Cover Points (CPs) for industry GPU units, thus enhancing the overall functional coverage process.

2.5 Test Generation using Constrained-Random Methods

Instead of simulating with random test vectors and expecting improved coverage, constrained-random test generation methods extract design-specific constraints to generate high-quality test vectors. Figure 12 provides an overview of the constrained-random test generation [15, 44, 47, 53, 61, 70, 86, 97]. The constraints and input biases are extracted from the specification and then tests are generated based on them. In order to measure the design coverage, monitors are injected into the designs. If the generated test vectors do not provide the expected coverage, more test vectors that adhere to the constraints are generated until the expected coverage target is achieved.

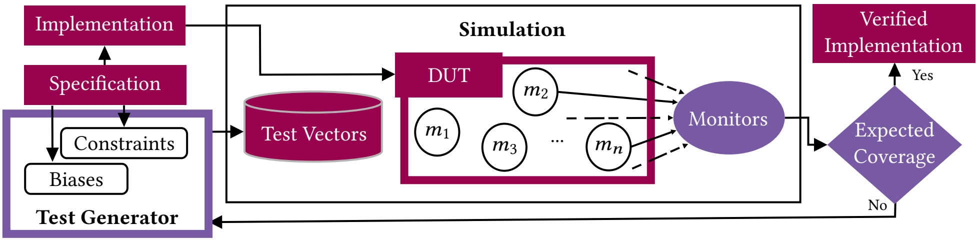


Fig. 12. An overview of constrained-random test generation.

Kitchen and Kuehlmann proposed a constraint specification technique for test generation with constrained-random methods [61]. Proposed technique can be elaborated with an example as follows, Let us consider m Boolean variables as $x = (x_1, \ldots.., x_m)$ and $y = (y_1, \ldots., y_n)$; $(-2^{B-1} \leq y_i \leq 2^{B-1} - 1)$ denote a vector of n variables such that B is a positive integer. Let the constraints on assignments to x denoted by $f(x)$, such that $f(x) = 1$ for all valid assignments of x. If we arrange constraints on y to be conditional on x, different assignments to x can trigger constraints on y values. Let $g^{(x)}(y)$ denotes the constraint which is active for a set of values of x. Then the valid test vector assignment can be represented by Equation 2.

$$\{(x, y) : f(x) = 1 \wedge g^{(x)}(y) = 1\} \tag{2}$$

Specifically, the authors showed that $g^{(x)}(y)$ could be represented as disjunctions of conjunctions of predicates of $g_{ij}^{(x)}$ of the y variables, as they illustrated in Equation 3.

$$g^{(x)}(y) = \bigwedge_i \bigvee_j g_{ij}^{(y)} \tag{3}$$

The authors implemented linear and multilinear constraints on y and sampled the constraints with a Boolean/Integer Constraints Normal Form (MBINF) sampler. Then, they proposed a hybrid constraint solver based on Markov-chain Monte Carlo methods. Experiments showed that the proposed method could generate test vectors significantly faster based on the Davis–Putnam–Logemann–Loveland (DPLL) algorithm and was more robust than Binary Decision Diagram (BDD)-based sampling.

Guzey and Wang proposed a directed test generation framework using automated constraint extraction [47]. The authors extracted constraints during simulation such that they improved the controllability of internal signals. Then the extracted constraints were enclosed into a constrained testbench to generate test vectors to control multiple signals simultaneously. The authors conducted experiments on OpenSparc benchmarks to obtain the initial dataset for mining. After extracting the constraints, they generated test vectors and achieved a success rate of 90%.

Naveh et al. proposed a knowledge base centric constraint extraction technique for the generation of directed tests [97]. The proposed knowledge base contained descriptions of the design, the expected behavior of the design, and design-specific expert knowledge. They carried out test generation in two stages. Initially, stream generation was controlled by a test template. Following this, they modeled the transactions in the specifications as a Constraint Satisfaction Problem (CSP). They derived the variables involved in the CSP from the transaction model available in the knowledge base.

A different approach for the generation of constrained-random test vectors is by incorporating fuzzing techniques. The fuzzing technique is the simple process of running test patterns generated with a particular feed and observing the output. Fuzzing is a very popular technique used in the software community [44, 86]. There have been promising research efforts in generating directed tests with fuzzing for hardware verification, as highlighted in several notable works [15, 53, 70]. Canakci et al. proposed DirectFuzz, a directed test generation technique utilizing graybox fuzzing [15]. In the proposed gray box fuzzing methodology, there were six stages: (i) setting up the total fuzzing duration and the initial seed corpus, (ii) selecting seeds from the corpus, (iii) pairing the current seeds with energy levels to determine how many new seeds should be generated from the current seeds, (iv) generating N new seeds by mutating the current seeds, (v) executing the seeds with the design under test, and (vi) analyzing the results from the executions to check if they improved the coverage.

2.6 Test Generation using ATPG Tools

Automated test pattern generation (ATPG) tools are widely used for manufacturing testing. These tools generate directed tests based on the fault models, such as stuck-at faults and bridging faults. There have been efforts to leverage the advantages of manufacturing testing in different domains. For instance, various researchers have utilized Automatic Test Pattern Generation (ATPG) methods and fault models for design validation (and vice versa) [23, 27, 41, 49, 58, 125].

Cruz et al. proposed an ATPG-based directed test generation technique for detecting malicious implants in hardware designs in their work [27]. Figure 13 illustrates the overview of the proposed approach. First, the authors identified the possible Trojan trigger conditions within the design. They then created a Scan replacement circuit to proceed with model checking. Subsequently, they derived security properties in such a way as to activate the equivalent signals and gates from the gate-level implementation. These generated properties, along with the design where the Scan circuit had been replaced, were then fed into a model-checking tool to generate constraints. All possible Trojan triggers were modeled as Stuck-at-faults, and directed to the ATPG tool along with the generated constraints. The output from the ATPG tool could then be used to activate Trojan trigger scenarios. The authors conducted experiments on various benchmarks, managing to outperform MERO [16] in terms of Trojan coverage.

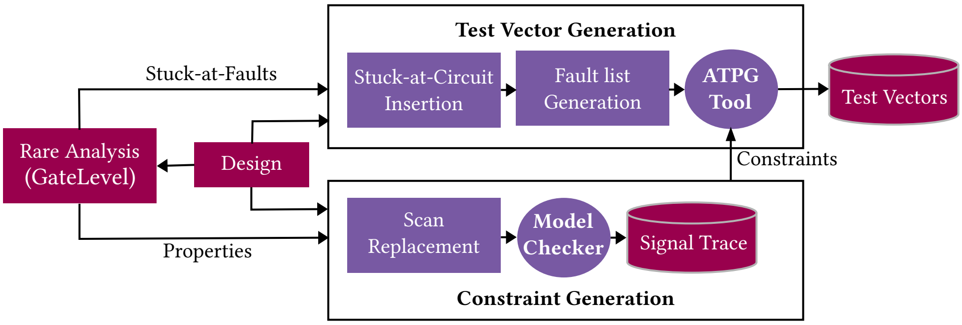


Fig. 13. Proposed constrained random test generation framework in [27]

Another directed test generation scheme utilizing ATPG tools for activation of Trojan triggers is presented in [125]. Initially, the authors explored the modeling of hardware Trojans and various types of potential Trojan implementations. Following this, a methodology was outlined for generating tests with the ATPG tool, specifically targeting the activation of Trojans with single trigger conditions. All single-trigger Trojans were modeled as stuck-at-faults and fed into the ATPG tool to obtain the test pattern. In the experimental section, the effectiveness of the proposed approach was demonstrated, showing its capability to deal with Trojans that have multiple trigger combinations.

Jayasena and Mishra proposed an ATPG-based test generation framework to activate rare events in hardware designs [58]. Instead of relying on bit flipping to generate new test cases, the proposed method generated new test vectors from the ATPG tools, forcing the tests to be valid and have maximum effectiveness in activating rare events. The authors demonstrated the effectiveness of ATPG-based N-activation and maximal clique activation of rare events, illustrating the effectiveness of complex hardware designs.

3 TEST TRANSLATION

Directed tests can be generated to verify various abstraction levels in the hardware design flow as discussed in Section 1.1. It is faster to generate tests at higher abstraction levels since it has less implementation-specific data. However, the generated tests should be translated (refined) to be applicable for other abstraction levels. In this section, we look at various research efforts to translate tests that were generated at higher abstraction levels in order to apply them at the lower abstraction levels.

3.1 High-Level to Low-Level Refinement of Tests

Chen et al. proposed a methodology to generate RTL test vectors using TLM specifications [20]. The proposed methodology consists of three major steps as illustrated in Figure 14. In the first step, SystemC TLMs were translated to SMV specifications. Then, following a fault model, a set of properties were derived to facilitate the test generation. Finally, the TLM tests were translated into RTL tests. The proposed TLM to RTL test translation procedures were independent of the test generation process. Therefore, the tests that were generated on TLM using other methods (or manually added later) could be translated into RTL utilizing the proposed test translation techniques. A major challenge in test translation is to bridge the gap between high-level and low-level abstraction levels. The authors utilized a set of rules (templates) for TLM-to-RTL test translation using a Test Refinement Specification (TRS) language. They ensured that any missing data required to fill the abstraction level gap was captured by the TRS.

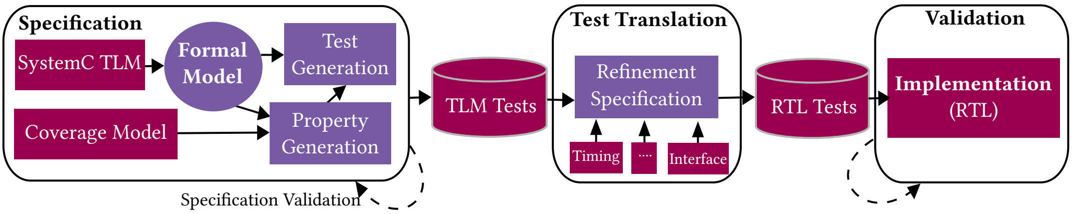


Fig. 14. Proposed test translation technique in [20]

.

Bombieri et al. proposed a method to generate test patterns at TLM level and translate them to RTL level [13]. The authors started with the specification and constructed the TLM abstraction. They then generated test vectors at the TLM level. Following this, they proposed an automated way to synthesize TLM test patterns into RTL test patterns, taking advantage of the structural information at the RTL level. These structural details were extracted during the RTL-to-TLM abstraction process. The test pattern translation process involved the concepts of TLM transaction mapping and the mapping of TLM transactions into RTL computational phases. Each TLM transaction generated a TLM test pattern, and this pattern was utilized by the TLM design under verification to elaborate and retrieve the corresponding result. The authors demonstrated that each of these TLM phases corresponded to an input, elaboration, and output sub-phase in the RTL design. To finalize the process, they introduced clocked transitions to form the RTL sub-phase of the test pattern. The authors achieved a significant speedup (ranging from 6X to 68X) on various designs compared with test generation on RTL models.

Farahmandi et al. proposed a test generation technique that facilitates post-silicon debugging using TLM models in [37]. Observability is a key property that is important for post-silicon debugging. The proposed approach is useful when a golden TLM design is available but RTL implementation may be buggy. Therefore, generating tests using a buggy design can lead to useless tests. In other words, the proposed approach aimed to create tests utilizing the golden (TLM) models while integrating assertions and observability constraints taken from the RTL models. The primary goals of the proposed methodologies were to transform RTL assertions (ϕ) and observability constraints (ψ) to create modified observability-aware assertions (π). Then, π needed to be mapped to TLM assertions (α) to generate TLM tests. Finally, the TLM tests were translated into RTL tests. The authors demonstrated that their proposed method significantly outperformed the use of random test generation techniques for bug detection.

3.2 Low-Level to High-Level Abstraction of Designs

There are promising approaches that have been explored to convert RTL designs to high-level abstractions with the aim of reducing the overall validation effort [14]. For example, fault simulation is a popular functional verification technique for RTL models. Bombieri et al. proposed techniques to speed up the process of fault injection and simulation on RTL models, utilizing TLM models [14]. Figure 15 presents the overview of the proposed methodology. They started with fault-injected RTL models. Next, fault-injected RTL models were abstracted into TLM models. Then, the test patterns were generated using TLM models, and fault coverage was recorded. The test pattern synthesis method proposed in [13] was utilized to translate the TLM tests to corresponding RTL tests. The authors demonstrated that the proposed technique was significantly faster than direct RTL fault-injection-based verification.

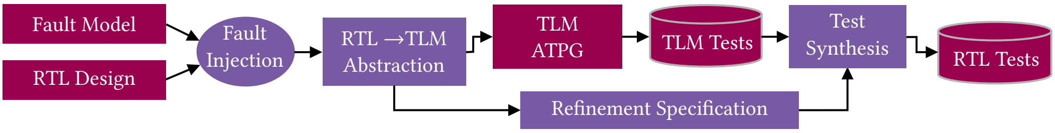


Fig. 15. Proposed design translation technique in [14].

4 HARDWARE VALIDATION USING DIRECTED TESTS

We described various test generation as well as test translation techniques in the previous sections. The generated test patterns can be used in a wide variety of application scenarios. In this section, we focus on three usage scenarios for directed tests: (i) pre-silicon functional validation, (ii) post-silicon validation and debugging, and (iv) validation of non-functional requirements.

4.1 Pre-silicon Functional Validation

Pre-silicon validation refers to the hardware verification effort prior to fabrication. For example, this involves functional validation of TLM, RTL, and gate-level designs. The verification engineers utilize different validation methods to ensure that the pre-silicon designs is bug-free prior to fabrication. The validation methods differ based on the coverage metric (e.g., code coverage), abstraction levels (e.g., RTL models), as well as design types (e.g., processor, memory, etc.). In this section, we describe pre-silicon validation methods in four categories: coverage-based validation, processor validation,, memory validation, and assertion-based validation.

4.1.1 ***Coverage-based Validation.***

One of the important aspects during design validation is how to determine the validation progress. Various coverage metrics, such as branch coverage and FSM coverage, are used during functional validation. Random test vectors can cover easy-to-detect scenarios. Coverage-directed functional verification focuses on the coverage of the designs with the directed test vectors. Specifically, it focuses on generating directed tests to activate hard-to-detect corner cases such as rare sequences of events, rare combinations of triggers and branches, complex and rare state transitions, etc.

In Section 2.2, we discussed different concolic testing methods. There were research efforts where concolic testing had been used to cover the corner cases in the designs [2, 3, 74, 76, 79, 107, 110]. The coverage-directed concolic testing worked well on control-intensive designs. A coverage-directed approach was necessary since control-intensive designs usually contained complex state machines that random and constrained-random test generation techniques might not have covered. Based on the coverage analysis, it was observed that the quality of the heuristic directly affected the coverage, and the complexity of computing the heuristic directly affected the scalability of the

Table 1. Summery of coverage directed test generation with concolic testing methods.

Framework	Heuristic	Coverage			Notes
		Low	Med	High	
[42]	Register Transitions	✓			Path explosion problem
[74]	Path Exploration		✓		Path explosion problem
[75]	Branch Coverage		✓		Path explosion problem
[76]	Symbolic State		✓		No path explosion problem
[110]	Trace Analysis		✓		Support dynamic array references
[2]	Basic Block Distance			✓	Activates specific targets
[3]	Branch Counter			✓	Uniform coverage
[84]	Target Clustering			✓	Transfer learning to next target

test generation framework. Table 1 illustrates the summary of the coverage results from different concolic testing methods discussed in Section 2.2.

4.1.2 *Processor Validation*.

Functional verification of processors is a major challenge due to the increasing complexity of modern processors. There were promising validation techniques proposed to overcome different inherent problems associated with the validation of processor architectures. Benjamin et al. conducted a functional verification case study on superscalar microprocessors using test cases generated with the help of formal methods [8]. In these case studies, the authors managed to achieve a 50% improvement in transition coverage with less than a third of the test vectors compared to implementation verification tests.

There were efficient test generation techniques for validating pipeline processors utilizing property decomposition combined with SAT-based bounded model checking [63, 65, 66]. The authors first extracted properties from the processor model. Next, they decomposed the original properties into the smallest possible sub-properties. Then, they utilized SAT-based bounded model checking for generating test vectors, as shown in Figure 6. The authors also discussed various issues related to composing the generated test vectors to construct the test vector corresponding to the original property.

Dang et al. presented a test generation technique to cover all possible processor pipeline scenarios [28]. The authors first defined a processor model based on communicating extended FSMs. They generated test vectors by covering all possible states and transitions. For test generation purposes, they first constructed the global FSM. Next, the test generation algorithm traversed through the global FSM while generating test vectors for individual paths. The proposed algorithm constructed the FSM and generated the test vectors on the fly without storing the global FSM in advance.

There were also approaches to generate test vectors based on processor specifications. Mishra and Dutt described a specification-driven directed test generation technique for validation of pipelined processors [89]. They utilized model checking-based test generation using the functional fault model and the graph model of the architecture. For each fault in the fault model, they generated a property. Next, the test vectors were generated from the property using model checking. The authors showed 100% coverage in different fault models of register read/write, operation execution, execution path, and pipeline execution.

The concept outlined above has been extended for the validation of multi-core architectures [109]. The test generation procedure followed the same steps outlined in [89], but the properties considered here were related to multi-core architectures. The authors derived properties that should adhere when several processor cores share the same communication bus and a shared memory subsystem. This approach enabled the reuse of the knowledge learned from one core to the remaining cores in multi-core architectures (structural symmetry), from one bound to the next for a given property (temporal symmetry), as well as from one property to other properties (spatial symmetry). Experimental results revealed that the proposed test generation framework outperformed existing multi-core validation techniques.

Table 2. Summary of different test generation techniques to validate processor designs.

Technique	Test generation Method	Notes
[8]	Formal methods	Imporved test compaction
[63, 65, 66]	Model checking	Focused on pipelined processors
[89]	Model checking	Use property decomposition
[28]	Constraint Random	Using FSM to construct tests
[109]	Model checking	Focused on multicore processors

Table 2 illustrates the summary of different processor validation techniques proposed in the literature. It can be observed that most of these methods rely on formal verification for test generation, such as SAT-based bounded model checking.

4.1.3 *Memory Validation*.

Memory and cache protocol validation is an important and expensive process during hardware validation [34, 84, 108]. Elver and Nagarajan presented a fast memory verification technique for a full system design implementation under simulation [34]. Usually, memory operations are non-deterministic and there are an exponential number of possible scenarios. The authors tackled both non-deterministic behavior and the exponential complexity by using Genetic Programming (GP). The proposed framework identified small memory operations that could happen separately. Using a crossover function, the authors prioritized the operations that led to non-determinism. As a result, the memory controller bugs got activated. Next, the design under test was simulated with the generated test vectors while observing the coverage of the design. The authors evaluated various memory designs using the gem5 cycle-accurate simulator [77]. The proposed method outperformed pseudo-random test generation on different memory designs.

While formal methods were explored for cache coherence validation [117, 124, 134], they either used a formal (abstracted) model of the protocol or could lead to state space explosion when dealing with complex RTL implementations. Simulation-based methods could handle memory designs at different abstraction levels. There were promising efforts for generating directed tests for the validation of cache coherence protocols [84, 108, 126]. Wagner and Bertacco designed the MCjammer that could achieve higher state coverage compared to constrained-random tests [126]. Qin and Mishra developed a directed test generation technique that analyzed the possible state space of the global FSM of the cache coherence protocol [108]. They developed graphical representations of the state space for several commonly used cache coherence protocols, which could be viewed as compositions of simple structures. Next, they presented an on-the-fly directed test generation algorithm based on Euler tour [32], which required linear space with respect to the number of cores [108]. This approach could reduce the number of tests by half compared to the tests generated by breadth-first search [108]. However, it could still be impractical to verify all possible transitions in the presence of a large number of cores. Lyu et al. extended the test generation method using quotient state space [84]. They tried to select important transitions by utilizing equivalence classes and omitted only similar transitions. Experimental results demonstrated an interesting trade-off between the transition coverage and validation effort of cache coherence protocols.

4.1.4 *Assertion-based Validation*.

Assertions are widely used in the industry to verify the functionality of hardware designs. Assertions can be viewed as checkers, implanted inside the design, that monitor certain functional behaviors. If the expected behavior is not observed, the assertion will get triggered with warnings or error messages. This message can lead to an easier debugging process and faster bug localization. There are two key challenges in assertion-based validation: (i) how to generate a set of important assertions, and (ii) how to ensure that the generation assertions are valid. Witharana et al. provided a survey of assertion generation techniques [131]. Directed test generation proved useful for checking the validity of the generated assertions. Specifically, it offered a framework to find counter-examples that violated the assertions [132]. Figure 16 provides an overview of directed test generation for activating assertions.

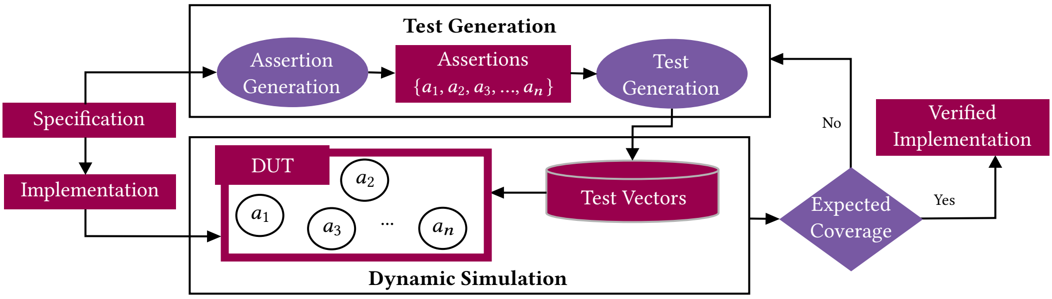


Fig. 16. Overview of directed test generation during assertion-based validation.

Directed test generation had been explored for assertion-based validation in diverse domains, including System-on-Chip (SoC) and Network-on-Chip (NoC) designs [56, 99]. Jayasena et al. presented a NoC validation framework using assertions [56]. The authors derived a set of assertions from the NoC specification, embedding them into the implementation as pre-silicon checkers. They validated the generated assertions by generating input vectors using bounded model checking as well as concolic testing. Oddos et al. proposed a directed test generation framework for assertion-based verification of SoC designs [99]. First, the assertions had been converted into PSL properties. Next, a set of checkers was constructed from each of the properties. There could be multiple paths that could satisfy the same property. In order to select one transition, a pseudo-random block was utilized. Finally, the test vectors were constructed using Boolean satisfiability solving.

Tong et al. presented an assertion-based directed test generation framework with test compaction capabilities [122]. The authors expected the assertions to be derived from the specification. Typically, assertions could be explicitly denoted using expected sequences of events (expressed in terms of temporal operators or regular expressions using Boolean expressions). The authors used assertions as a coverage metric for the design under verification. For designs that didn't have internal access (such as IPs from third parties), the authors generated assertions by only referring to the design input and output interface signals.

Assertion-based validation was also promising for checking the security and trustworthiness of SoC designs [132]. In this approach, the assertions were derived based on structural and dynamic analysis of designs. The authors outlined eight classes of vulnerabilities and generated assertions to monitor the implementation for any security violation. The authors utilized concolic testing to activate these assertions. Specifically, the authors first identified the assertion type and then rewrote the assertions using branch statements. Next, they considered the generated branch as the target for the concolic testing to activate. Once the concolic testing generates the test pattern, the generated test could be used to activate the security assertion. The authors showed that the generated test patterns could achieve 100% assertion coverage in diverse designs.

4.2 Post-silicon Validation and Debug

In spite of extensive efforts during pre-silicon validation, it is not feasible to capture all functional bugs as well as electrical errors. The objective of post-silicon validation is to detect and fix these escaped bugs. Since we are dealing with integrated circuits (post fabrication), a major challenge during post-silicon debug is the observability of the internal signals [90, 91]. Therefore, the test generation techniques have to ensure that the effect of the generated test can be viewed at observable points, such as trace signals or primary outputs. In this section, we survey directed test generation methods in four application scenarios: validation of functional behaviors, error detection, fault localization, and validation of soft errors.

4.2.1 *Post-Silicon Validation of Functional Behaviors*.

There are many efforts for functional validation of hardware components at the post-silicon stage [1, 9, 26, 62, 103, 104, 116, 127]. In previous sections, we discussed pre-silicon validation and using concolic testing based test generation. Cong et al. proposed a concolic testing framework for post-silicon validation [26]. In a typical ASIC design flow, post-silicon tests are prepared before the silicon prototype (integrated circuit) becomes available. Therefore, the proposed concolic testing framework generated post-silicon test cases with virtual prototypes. The two major steps (concrete simulation and symbolic execution) were identical to the corresponding steps discussed in Section 2.2, except that they were applied to the virtual prototype implementation. The authors took into consideration device transactions to generate the test vectors. They used the QEMU emulation environment to emulate the virtual prototypes and generate test cases. Figure 17 illustrates the major steps involved in the proposed test generation algorithm.

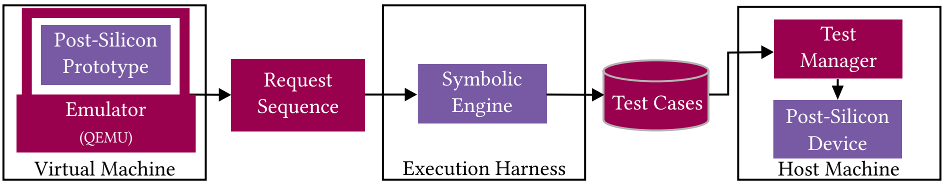


Fig. 17. Concolic testing for post-silicon validation [26]

Wagner and Bertacco proposed a post-silicon processor validation technique Reversi [127] using test generation. The author's main objective was to maximize the performance potential of silicon prototypes and bypass the costly simulation step typically needed for post-silicon validation. Reversi generated tests that restored the initial state of the machine upon execution completion, thereby eliminating the need for a simulation phase. The absence of a simulation step allowed for direct test generation by hardware on the system board, reducing the need for expensive test-generation servers, and thereby accelerating the testing process by 20x.

Threadmill [1], developed as a bare-metal, user-directable exerciser with a simple pseudo-random test generation engine, aimed to support a unified post-silicon verification methodology. Threadmill operated as an on-platform exerciser, continuously generating test cases [62], executing, and checking tests. The proposed framework comprised an OS-like layer, data structures representing the test template and model, and a fixed code for exercising the hardware. With a focus on multi-threaded designs, Threadmill generated multi-threaded test cases, and authors have demonstrated the effectiveness of Threadmill on the POWER7 processor chip.

An overview of test generation techniques for identifying escaped bugs from the pre-silicon stage was discussed in [9]. The authors discussed the challenges in post-silicon validation and elaborated on the usage of Reversi [127] for processor core validation combined with the CoSMA architecture for the validation of the memory subsystem. CoSMA was used to observe tests executing on the testing platform and ensure data sharing adhered to the coherence protocol in use. Experimental findings indicated that it enabled high coverage verification with minimal performance overhead (less than 23%) and negligible area impact (approximately 1%).

Papadimitriou et al. presented a comprehensive set of bug models for address translation mechanisms (ATM), categorizing the impacts of functional errors and electrical bugs within the hardware structures employed in address translation [103]. Then, the authors proposed a self-checking, ISA-independent constraint random test generation methodology, which, when evaluated using the developed bug models, minimized bug detection latency and maximized the utilization of the silicon's performance to achieve improved validation coverage. This concept was extended to evaluate address translation caching arrays at the post-silicon stage [104]. The authors were able to detect and achieve 100% post-silicon bug coverage in a simulation environment.

4.2.2 ***Detection of Post-Silicon Errors.***

Error detection latency is a major consideration during post-silicon verification. The time elapsed from the occurrence of an error to the detection of the error is known as the error detection latency. Hong et al. proposed a Quick Error Detection (QED) technique by transforming existing post-silicon test vectors to drastically reduce the error detection latency [51]. The proposed QED transformation technique allowed tradeoffs between coverage, error detection latency, and complexity. This technique focused on electrical bugs since the detection of electrical bugs was more time-consuming, and electrical bugs could be modeled as bit-flipping at flip-flops. The authors conducted experiments with existing test vectors based on the ideas presented in fault-tolerant computing [78, 85, 100]. The proposed QED technique considered the following four factors: (i) post-silicon validation tests did not need to care about containment and recovery, (ii) performance penalties introduced during test transformation might be acceptable in post-silicon validation, (iii) post-silicon test vectors might be developed at pre-silicon stages, and they could be optimized by QED transformation, and (iv) QED transformation of test vectors should not affect the coverage. The authors presented two complementary ways for the QED test transformation: (i) error detection by duplicate instruction for validation, and (ii) redundant multi-threading for validation where both were extensions from fault-tolerant computing. The authors claimed that the proposed QED could drastically reduce the error detection latency with pre-generated test vectors.

Lin et al. proposed a quick error detection technique for identifying hard-to-detect post-silicon bugs in hardware designs [73]. The test generation scheme was capable of quickly detecting bugs in processor cores, cache controllers, memory controllers, network-on-chip components, and multi-core network-on-chips during post-silicon validation. The proposed QED technique shortened the error detection time from billions of cycles to several hundreds of cycles for most of the bug scenarios. Similar to [51], the authors assumed that the initial set of test vectors were given to the framework. Next, they utilized a statistical technique to transform given test vectors to overcome the detection latency problem. The proposed statistical transformation used spatial Proactive Load and Check (PLC) transformation. The difference between the proposed technique and [51] was that the PLC transformation technique did not depend on the re-execution of instructions from the original test. Instead, it modified the targeted instructions by adding PLC operations at finer granularity across memory and input/output (IO). The main objective was to strategically perform loads with all the threads present in all the processors from a selected set of variables while adding self-consistency checks on those selected variables. As a result, the process enabled the detection of bugs present in the processor cores within reduced detection latency.

4.2.3 ***Localization of Post-Silicon Faults.***

While there are promising efforts to detect design faults in hardware implementations, localizing those faults is an important task. There were several promising efforts for facilitating fault localization with directed test generation. A mutation-based fault detection and localization technique was presented [29]. Figure 18 illustrates the steps involved in the proposed test generation scheme. Proposed procedures started with a hardware design with a set of initial failing patterns. The authors first generated an initial set of suspect statements utilizing effect clause analysis. Then, alternate mutant statements were generated such that there were more mutant statements for each suspect statement and fewer mutants for each non-suspected statement. From each generated mutant statement, a diagnostic test pattern was then derived utilizing bounded model checking, such that the generated test pattern was able to distinguish the mutated design from the original design with error. The authors showed that ATPG-based mutation methods could localize faults efficiently compared with existing methods.

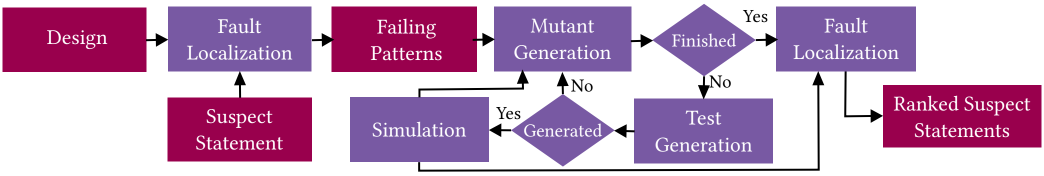


Fig. 18. Fault localization using directed tests [29]

A framework for directed test generation with the ability of automated debugging was presented [36]. Figure 19 presents the major steps involved in the proposed test generation process. The authors demonstrated that their proposed technique was capable of activating any bugs present in arithmetic circuits using the generated test vectors. They first ran equivalence checking on the implementation against the specification, representing both as polynomials. If the specification and implementation were equivalent, the remainder would be zero. Conversely, a non-zero remainder implied that the implementation was buggy. They then attempted to find the input assignment that would make the remainder non-zero. There could be multiple assignments capable of making the remainder non-zero, each of which could activate a bug. Since the test vectors were available, they could be applied to the implementation to find the existing bug. To expedite bug localization, the authors identified the regions that produced faulty outputs and their intersections with the generated test vectors. Automated debugging was enabled by identifying specific patterns in the remainder, and the authors showcased the effectiveness of their approach using various arithmetic circuits.

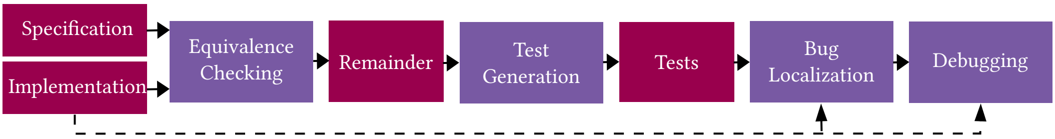


Fig. 19. Directed test generation for faster bug localization [36]

DeOrio et al. proposed a post-silicon bug localization technique called BPS [30] ("Bug Positioning System"). It employed a two-part approach, initially logging compact signal activity observations using an on-chip hardware component and subsequently conducting off-chip software post-analysis. These observations were then condensed into a compact encoding for various test executions, with slight differences between passing and failing runs. The authors used a statistical approach to analyze collected data to identify variations in signal activity caused by bugs. As a result, BPS could pinpoint the approximate clock cycle and the set of signals closely linked to the detected error, and the proposed technique could be applied to existing hardware designs with less than 1% of area overhead.

4.2.4 *Post-Silicon Validation of Soft Errors.*

In addition to detecting the functional errors escaped during pre-silicon validation, post-silicon debug needs to detect soft errors, crosstalk faults, and glitches [31]. If the magnitude of this glitch is sufficient enough, this can flip the internal bit values of the design and may cause incorrect functional output. This phenomenon is also known as single event transient. If the effect of the bit flip is propagated to the design output, then it is called a single event upset. The framework proposed by Basu et al. attempted to generate test vectors to detect all the single-event transient situations that might have had the possibility of affecting the functional output of the design [7]. The authors utilized stuck-at-fault modeling for soft-errors. The authors generated the fault list by analyzing the design. They then performed signal selection aware test generation and test generation aware signal selection. The overview of the proposed approach is presented in Figure 20. It illustrates the proposed procedure of test generation using the ATPG tool followed by signal selection and trace selection. The authors conducted experiments with ISCAS benchmarks and achieved a 58% improvement in detecting crosstalk faults compared to profile-based signal selection.

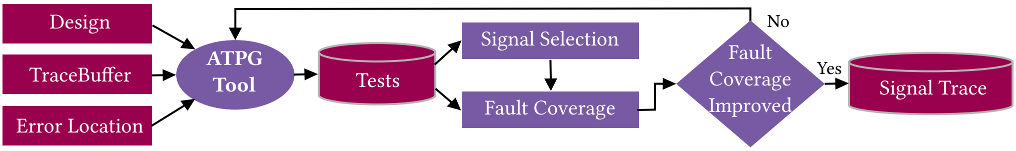


Fig. 20. Observability-aware test generation for detecting electrical errors [7]

Foutris et al. proposed an approach that aimed to identify failures in silicon prototypes, which might signify potential silicon bugs [39]. A key feature of the methodology was the generation of enhanced random instruction tests, enabling the detection of design bugs without the need for golden responses. A lightweight hardware mechanism was proposed to record mismatches between the results of two equivalent instructions, facilitating the identification of the offending instruction and reducing the validation data forwarded to the debug process. The authors conducted experiments and illustrated the effectiveness of the proposed methodology by identifying all injected bugs. Although general solutions like QED [51] prove effective in the post-silicon validation process, complex implementations require special attention as illustrated by research efforts specifically focused on sub-components [36, 103, 104].

4.3 Security and Trust Validation

Given the growing complexity of modern systems, there are potential vulnerabilities that can be exploited in hardware implementations. This section surveys various test generation-based techniques for security and trust validation.

*4.3.1 **Trust Validation with Detection of Malicious Implants**.*

Malicious implants, popularly known as hardware Trojans, the threat model assumes that an attacker can insert malicious implants such that they can be triggered by extremely rare conditions, which can stay hidden during normal functional validation. The insertion of hardware Trojans can happen at various phases in the supply chain [87]. In this section, we focus on proposed test generation techniques to activate the stealthy triggers [16, 27, 52, 82, 96, 101]. Figure 21 illustrates the steps involved in proposed hardware Trojan detecting techniques under this threat model. First, the designs were simulated with thousands of random test vectors. Next, the suspicious (e.g., rare) signals were calculated by observing the signal values during the simulation. The following step was directed test generation with the objective of maximizing the likelihood of activating the trigger conditions. To evaluate the quality of the generated test vectors, Trojans were injected into the designs and simulated with the generated test vectors to observe the Trojan coverage.

As discussed in Section 2.3, MERO [16] utilized the N-detect principle [105] to generate test patterns that were likely to activate the trigger if the N was sufficiently large. Saha et al. improved

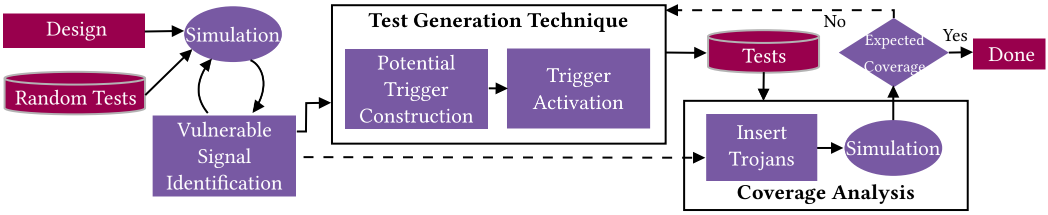


Fig. 21. Overview of hardware Trojan detection techniques proposed in the literature.

MERO [16] using a genetic algorithm and Boolean satisfiability (Section 2.1) [114]. The authors provided the negation of the Trojan trigger condition to the SAT solver, and the relevant test pattern to activate the trigger was generated by the SAT solver. Further, the authors proposed a test compaction technique to select the test vectors that would activate the most sampled trigger conditions using genetic algorithms. Pan and Mishra presented a test generation framework that outperformed existing methods [16, 114] using reinforcement learning [101] (Section 2.4).

Banga and Hsiao presented a region-based Trojan detection concept [6]. The proposed approach consisted of two stages. First, they identified appropriate regions from the circuit (region-based partitioning), and then they derived input test vectors to maximize the relative power consumed by each selected region. The determination of regions was done by flip-flop radius. For a one flip-flop radius, it contained one flip-flop and all the transitive fan-in and fan-out gates. All the regions were within the clock boundaries. Once they identified the regions, the authors created activity peaks on a per-region basis. They simulated the design with test vectors and isolated the test vectors to maximize the activity in a given region while minimizing the switching activity in other regions.

Lesperance et al. presented a Trojan detection approach by exhaustively searching through a k-bit subspace [71]. The authors demonstrated that in cryptographic IPs, n plain text bits are ideal Trojan triggers for attackers. Since the plain text was an input to the system, each bit would have similar observability and controllability values, which made the existing Trojan detection algorithms fail during the initial filtering of rare signals. The proposed test generation approach assumed that the trigger width was k and attempted to trigger all possible combinations of k out of n inputs. Furthermore, the authors proposed heuristics to determine k and validated the proposed technique on an AES cryptographic core.

Sabri et al. presented a SAT-based test pattern generation technique with path delay analysis for the detection of combinational Trojans [113]. The authors employed MUX-based debugging techniques to localize the problem with generated test patterns. They generated test cases using an automated SAT-based test generation scheme with path delay analysis. To measure the accurate path delay, the authors utilized a clock-sweeping technique. For the test generation, potential trigger signals were determined by sequentially sticking each net to '1' and '0'. The authors demonstrated a Trojan coverage resolution of around 99.6% with a Trojan localization resolution of 99.6%.

SAT-based test generation and ATPG were utilized for detecting malicious implants [27, 82]. Lyu et al. utilized SAT solving to identify clusters of rare signals in hardware designs. They then used these clusters to find the largest clique, and employing SAT solvers, they generated the test vectors to activate the largest clique of rare signals in the design. On the other hand, Cruz et al. utilized model checking to generate constraints for activating rare signals in hardware designs [27]. The outputs from model checking were transformed into constraint structures for ATPG tools, facilitating the generation of test vectors to activate rare signals using ATPG tools.

Methods that were discussed so far require a golden model for security validation. Directed tests from these methods are provided as the inputs to the golden model as well as the suspicious implementation, and then outputs are compared against each other. If there is any malicious implant in the implementation, the output will differ. A practical problem arises when we do not have the golden reference model (the designer only has the implementation). Narasimhan et al.

proposed a two-step, golden design-free methodology for detecting malicious implants [96]. Instead of comparing the output with a golden design, the proposed methodology involved comparing the side-channel signatures of similar components within the implementation. First, they decomposed the design into components with similar structures. For each module, they utilized MERO [16] to generate tests. Finally, they applied the generated tests to self-similar components (e.g., two identical ALUs in a processor) and compared the respective side-channel signatures.

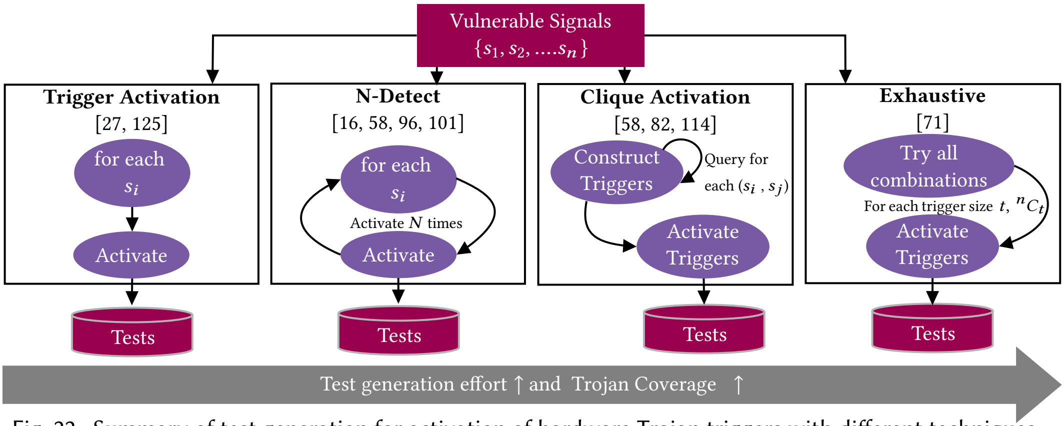


Fig. 22. Summary of test generation for activation of hardware Trojan triggers with different techniques.

Figure 22 summarizes different test generation techniques for Trojan detection. All these methods considered suspicious (rare) signals in the designs. They then utilized different test generation techniques (as discussed in Section 2) to activate Trojan triggers. As illustrated in Figure 22, there were methods that aimed to activate each suspicious signal once, N times, or even attempted to activate all possible Trojan triggers at once. Note that while the Trojan coverage improved, the test generation effort also significantly increased when the discussed methods attempted to activate all possible Trojan triggers.

4.3.2 *Security Validation with Side-Channel Analysis*.

There were promising research efforts that tried to maximize the side-channel sensitivity for improved Trojan detection. For example, they generated test vectors to maximize switching in suspicious regions while minimizing switching in the rest of the design [52, 83, 96, 98].

Narasimhan et al. improved the idea proposed in [96] by introducing more side-channel parameters into consideration [95]. Dynamic currents and maximum operating frequencies have an intrinsic relationship and this relationship was considered when introducing more side-channel parameters. In addition to the current signature which had been considered in [96], the authors considered quiescent currents, dynamics, and leakage currents which could directly affect the process variations with an implanted Trojan.

A directed test generation framework using delay-based side-channel analysis was presented [81]. The proposed technique was composed of three important components. First, the authors proposed a test generation scheme by changing the critical path to maximize the observable path delay. Next, the authors utilized the SAT-based test generation algorithm to generate test patterns for delay-based side-channel analysis. Finally, the authors proposed a hamming-distance-based test reordering technique to reorder the generated tests. Here, a distance evaluation method was utilized to increase the likelihood of building the critical path from the Trojan trigger to the payload.

To improve the delay-based side-channel analysis, Pan et al. presented a directed test generation framework using delay-based side-channel analysis for Trojan detection [102]. The primary intention of the proposed method was to generate a set of test vectors such that for every consecutive pair of vectors, the delay-based side-channel sensitivity was maximized. The authors first generated an initial set of test vectors using SAT solving. Next, the authors used a reinforcement learning agent trained with a stochastic learning scheme to increase the probability of activating rare signals. The machine learning algorithm continued to optimize the test vectors to sensitize the path delay iteratively until it generated the desired test patterns.

Table 3 summarizes different side-channel parameters considered for hardware Trojan detection using directed test generation.

Table 3. Summary of side-channel based Trojan detection with directed test generation

Technique	**Side-Channel parameter**				**Test generation method**	**Trojan Coverage**
	Path delay	*Power*	*Current*			
			Dynamic	*Leakage*		
[96]			✓		Statistical	medium
[95]			✓	✓	Statistical	medium
[52]		✓	✓		Statistical	medium
[80]			✓		Statistical	medium
[118]			✓		Statistical	medium
[83]		✓	✓		Statistical	high
[98]		✓	✓		Statistical	high
[102]	✓				Machine Learning	high
[81]	✓				Formal methods	high
[113]	✓				Formal methods	high

4.3.3 *Test Vector Leakage Assessment on Cryptographic Implementations*.

The concept of test vector leakage assessment (TVLA) for hardware implementations aims to ensure that the execution of an implementation does not expose confidential information through power side-channel signals at the early stages of the design lifecycle. There were various proposed techniques to perform test vector leakage assessment at the pre-silicon stage [50, 55, 106, 135]. TVLA process involves several key stages, as illustrated in Figure 23. The initial step is a hamming distance or hamming weight-based test generation phase with the intention of inducing power signature discrepancies. Subsequently, the design undergoes simulation with generated inputs (such as key pairs and a fixed plaintext), from which the power signature is constructed utilizing the value change dump from the simulation. Statistical techniques like the *t-test and KL-divergence* are then applied to calculate the distinction between two power signatures. Ultimately, the implementation is classified as secure or vulnerable to side-channel threats based on a predefined threshold.

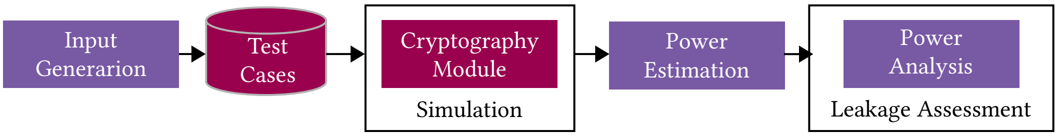


Fig. 23. Major steps of Test vector leakage assessment of pre-silicon hardware cryptographic implementations.

A framework designed for the automatic assessment of power side-channel vulnerability at the RTL level was presented in [50]. The framework conducted comprehensive analysis considering inter-block/module relations and offered fast power estimation based on RTL transition counts. With an average evaluation time of 35 minutes on different symmetric cryptography implementations, the framework provided hardware designers with ample flexibility to address PSC vulnerabilities at the RTL level. Zhang et al. introduced the PSC-TG, a framework designed for efficient and precise assessment of power side-channel leakage (PSCL) at the RTL level [135]. The proposed method consists of an RTL-level information tracking methodology for tracking the information flow of inputs. The authors generated test patterns for maximum PSCL using bounded model checking and introduced a new metric called SCV score to quantify pre-silicon PSCL using simulation power profiles. The authors showed the effectiveness of the proposed side-channel evaluation technique using several asymmetric key algorithm hardware implementations. As an extension to [50], Pundir et al. proposed a test vector leakage assessment that supported post-quantum supported hardware implementations of symmetric key algorithms [106]. Further, the authors proposed mitigations that could be applied to implementations that were vulnerable to power side-channel leakage.

Power side-channel evaluation of hardware implementations of asymmetric key cryptosystems requires a completely different methodology for performing test vector leakage assessment. Jayasena et al. proposed a test vector leakage assessment technique TVLA* [55], which primarily targets public key cryptography implementations. The proposed technique utilized constraint random test generation to generate input pairs and performed evaluations using statistical techniques. The authors performed experiments on hardware implementations of elliptic curve cryptography modules and RSA and were able to detect power side-channel vulnerable implementations at the pre-silicon stage.

Table 4. Summary of proposed techniques to perform test vector leakage assessment on hardware implementations of cryptographic algorithms at the pre-silicon stage.

TVLA	Cryptosystem	Technique	Evaluarted on	Features
[50]	Symmetric	KL-Divergence	AES	Flexibility
[135]	Symmetric	Welch's t-test	AES	Information flow tracking
[106]	Symmetric	KL-Divergence	AES	Countermeasures
[55]	Asymmetric	Welch's t-test	ECC, RSA	Countermeasures

The contributions of various proposed techniques for test vector leakage assessment are summarized in Table 4. It becomes apparent that different cryptographic implementations demand specific analysis methods, depending on the collisions that might streamline the key recovery procedure.

4.4 Validation of Non-Functional Requirements

The test generation techniques discussed in the previous sections focus on the validation of expected (functional) behaviors. In this section, we review test generation methods for validation of non-functional requirements, such as timing, energy, and temperature constraints.

4.4.1 *Validation of Timing Constraints.*

Krishnamachary et al. developed a directed test generation approach to validate the timing and delay of a design by utilizing the design hierarchy [68]. Usually, timing verification is done at the module level. The critical path of a module may get changed when it is integrated with the final system. Therefore, the authors first determined the critical paths inside the fully integrated design using fault injection. Next, they tried to generate test vectors to activate stuck-at-fault in the critical paths of the module. Since ATPG tools were not capable of handling complex hierarchical designs, the authors utilized a hierarchical test generation approach where all non-relevant modules were abstracted to simplify the process. After the critical paths were identified, the relevant tests were generated such that they could be applied at different implementation levels. The results showed that most of the module-level critical paths were no longer critical paths at the chip level.

Savir discussed delay test generation techniques for hardware designs [115]. This research work summarized different generator circuits to be used for delay-based test generation. Using plain linear feedback shift registers, plain cellular automata (a cascade of spatial type of FSMs), delay tests using two independent generator circuits, double-size generator circuits, input separation, and pseudo input separation were some of the techniques discussed related to delay-based test generation. The authors conducted extensive experiments to identify which techniques were beneficial for different instances.

4.4.2 *Validation of Temperature and Energy Constraints.*

There were promising attempts to validate temperature and energy consumption using directed test generation techniques [111, 130]. The proposed methods tried to find suitable processor frequency assignment test cases using formal methods on extended timed automata of the thermal and energy constrained multitasking environment. Validating a design against its thermal and energy budget is a challenging task due to the possibility of considering too many paths during the execution. This leads to path explosion problem with regular model-checking methods. Wang et al. calculated the extended timed automata for a particular configuration by the valuation of clock variables [130]. This resulted in a sequence of states from the source to the destination. Next, the authors generated properties in Computation Tree Logic (CTL) based on the temperature and energy budget. The test cases were generated using symbolic model checking. Due to the exponential number of states, model checkers tended to fail. The authors eliminated the path explosion problem by converting the problem into a Pseudo-Boolean Satisfiability problem. Qin et al. solved the path explosion problem by using an approximation algorithm [111]. The proposed approximation algorithm tried to generate a table for each state which contained information about the time consumption of all execution paths from that state. This value was proportional to the energy consumption and temperature value. Then, the authors utilized dynamic programming to approximately calculate the execution path and generate the test cases based on the calculated path.

Test generation is also promising for interconnect and voltage regulator design as well as from cooling perspectives. Since peak power determines interconnect dimensions and voltage regulator configurations, it is important to find the worst-case peak power scenario, popularly known as the power virus. A naive approach is to try all possible inputs to find the input sequence that creates the power virus. Unfortunately, finding such a sequence is infeasible due to an exponential number of input variations. There were research efforts to efficiently activate the power virus using test pattern generation [35, 40, 48, 67, 128, 129]. The common technique followed by these methods was that they sorted the circuit gates based on their fanout (output capacitance) and generated a test vector such that it assigned a transition to the gate with the highest fanout [35, 94, 128, 129]. Hajimiri et al. showed that it is required to consider the fanin cones of the gates which might block beneficial transitions in future iterations [48]. Therefore, in addition to sorting the gates by fanouts, the authors considered the potential negative impact of selecting a gate that could hinder transitions for other gates in subsequent iterations. The authors illustrated the effectiveness of the proposed approach with a significant 64% increase in switching activity compared to existing power virus generation techniques.

5 CONCLUSION

Hardware design complexity is increasing rapidly over the years. Efficient, fast, and affordable design verification techniques can save a considerable amount of design and validation effort by eliminating functional and non-functional bugs from hardware designs. Simulation based validation using random test patterns can capture a vast majority of easy-to-detect bugs. A major challenge is how to detect the remaining bugs (uncovered scenarios). Manual development of directed tests can be time consuming, error prone, and infeasible for large designs. Automated generation of directed tests is ideal for activating hard-to-detect bugs and corner case scenarios. In this paper, we

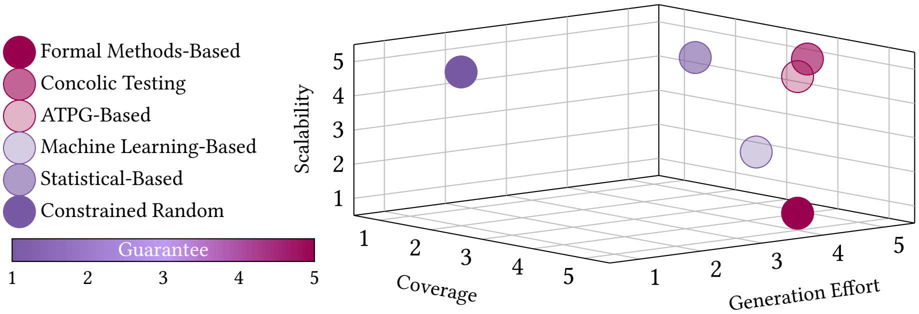


Fig. 24. Summary of different directed test generation techniques with respect to scalability, coverage, test generation time, and correctness guarantee (mathematical confidence).

presented a comprehensive survey of different techniques for directed test generation. Figure 24 provides a summary of different test generation techniques with respect to their scalability, coverage, effort taken to generate tests, and validation guarantee. We have also reviewed promising test translation techniques across different abstraction levels. Finally, we explored the use of directed tests in different validation scenarios, such as pre-silicon validation, post-silicon validation, security validation, and validation of non-functional requirements.

Despite their demonstrated potential, directed test generation remains underutilized in the semiconductor industry for various reasons. (1) There are too many directed test generation approaches, such as using model checking, bounded model checking, satisfiability solving, and concolic testing. (2) There are also different application domains with diverse requirements, such as observability-aware test generation in case of post-silicon validation, testing of functional behaviors versus security vulnerabilities, testing the validity of assertions, etc. (3) The traditional industrial frameworks rely heavily on hardware validation using random and constrained-random tests due to the inherent simplicity of the test generation methods. The future research and development efforts should focus on enhancing the user-friendliness and interoperability of hardware test generation frameworks, thereby facilitating seamless integration into diverse industrial environments.

ACKNOWLEDGMENTS

This work was partially supported by National Science Foundation (NSF) grant CCF-1908131.

REFERENCES

[1] Allon Adir, Maxim Golubev, Shimon Landa, Amir Nahir, Gil Shurek, Vitali Sokhin, and Avi Ziv. 2011. Threadmill: A post-silicon exerciser for multi-threaded processors. In *Proceedings of the 48th Design Automation Conference*. 860–865.

[2] Alif Ahmed, Farimah Farahmandi, and Prabhat Mishra. 2018. Directed test generation using concolic testing on RTL models. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, IEEE, 1538–1543.

[3] Alif Ahmed and Prabhat Mishra. 2017. QUEBS: Qualifying event based search in concolic testing for validation of RTL models. In *2017 IEEE International Conference on Computer Design (ICCD)*. IEEE, IEEE, 185–192.

[4] Tashfia Alam, Zhenkun Yang, Bo Chen, Nicholas Armour, and Sandip Ray. 2022. Firver: concolic testing for systematic validation of firmware binaries. In *2022 27th Asia and South Pacific Design Automation Conference*. IEEE, 352–357.

[5] Peter J Ashenden. 2010. *The designer's guide to VHDL*. Morgan kaufmann.

[6] Mainak Banga and Michael S Hsiao. 2008. A region based approach for the identification of hardware Trojans. In *2008 IEEE International Workshop on Hardware-Oriented Security and Trust*. IEEE, IEEE, 40–47.

[7] Kanad Basu, Prabhat Mishra, and Priyadarsan Patra. 2013. Observability-aware directed test generation for soft errors and crosstalk faults. In *2013 26th International Conference on VLSI Design and 2013 12th International Conference on Embedded Systems*. IEEE, IEEE, 291–296.

[8] Mike Benjamin, Daniel Geist, Alan Hartman, Gerard Mas, Ralph Smeets, and Yaron Wolfsthal. 1999. A study in coverage-driven test generation. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*. ACM/IEEE, 970–975.

[9] Valeria Bertacco. 2010. Post-silicon debugging for multi-core designs. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 255–258.

[10] Jayanta Bhadra, Magdy S Abadir, Li-C Wang, and Sandip Ray. 2007. A survey of hybrid techniques for functional verification. *IEEE Design & Test of Computers* 24, 02 (2007), 112–122.

[11] Jayanta Bhadra, Narayanan Krishnamurthy, and Magdy S Abadir. 2004. Enhanced equivalence checking: toward a solidarity of functional verification and manufacturing test generation. *IEEE design & test of computers* 21, 06 (2004), 494–502.

[12] K Uday Bhaskar, M Prasanth, G Chandramouli, and VAKV Kamakoti. 2005. A universal random test generator for functional verification of microprocessors and system-on-chip. In *18th International Conference on VLSI Design held jointly with 4th International Conference on Embedded Systems Design*. IEEE, IEEE, 207–212.

[13] Nicola Bombieri, Franco Fummi, and Valerio Guarnieri. 2011. Accelerating RTL fault simulation through RTL-to-TLM abstraction. In *2011 Sixteenth IEEE European Test Symposium*. IEEE, IEEE, 117–122.

[14] Nicola Bombieri, Franco Fummi, and Valerio Guarnieri. 2012. FAST: An RTL fault simulation framework based on RTL-To-TLM abstraction. *Journal of Electronic Testing* 28, 4 (2012), 495–510.

[15] Sadullah Canakci, Leila Delshadtehrani, Furkan Eris, Michael Bedford Taylor, Manuel Egele, and Ajay Joshi. 2021. Directfuzz: Automated test generation for rtl designs using directed graybox fuzzing. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, IEEE, 529–534.

[16] Rajat Subhra Chakraborty, Francis Wolff, Somnath Paul, Christos Papachristou, and Swarup Bhunia. 2009. MERO: A statistical approach for hardware Trojan detection. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, Springer, 396–410.

[17] Bo Chen, Kai Cong, Zhenkun Yang, Qin Wang, Jialu Wang, Li Lei, and Fei Xie. 2019. End-to-end concolic testing for hardware/software co-validation. In *2019 IEEE International Conference on Embedded Software and Systems (ICESS)*. IEEE, 1–8.

[18] Mingsong Chen and Prabhat Mishra. 2011. Decision ordering based property decomposition for functional test generation. In *2011 Design, Automation & Test in Europe*. IEEE, IEEE, 1–6.

[19] Mingsong Chen, Prabhat Mishra, and Dhrubajyoti Kalita. 2008. Coverage-driven automatic test generation for UML activity diagrams. In *Proceedings of the 18th ACM Great Lakes symposium on VLSI*. ACM, 139–142.

[20] Mingsong Chen, Prabhat Mishra, and Dhrubajyoti Kalita. 2012. Automatic RTL test generation from SystemC TLM specifications. *ACM Transactions on Embedded Computing Systems (TECS)* 11, 2 (2012), 1–25.

[21] Mingsong Chen, Xiaoke Qin, Heon-Mo Koo, and Prabhat Mishra. 2012. *System-level validation: high-level modeling and directed test generation techniques*. Springer Science & Business Media.

[22] Mingsong Chen, Xiaoke Qin, and Prabhat Mishra. 2010. Efficient decision ordering techniques for SAT-based test generation. In *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*. IEEE, IEEE, 490–495.

[23] Kwang-Ting Tim Cheng. 2003. The confluence of manufacturing test and design validation. In *International Test Conference, 2003. Proceedings. ITC 2003*. IEEE Computer Society, 1293–1293.

[24] Edmund Clarke, Armin Biere, Richard Raimi, and Yunshan Zhu. 2001. Bounded model checking using satisfiability solving. *Formal methods in system design* 19, 1 (2001), 7–34.

[25] Edmund M Clarke and Jeannette M Wing. 1996. Formal methods: State of the art and future directions. *ACM Computing Surveys (CSUR)* 28, 4 (1996), 626–643.

[26] Kai Cong, Fei Xie, and Li Lei. 2013. Automatic concolic test generation with virtual prototypes for post-silicon validation. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, IEEE, 303–310.

[27] Jonathan Cruz, Farimah Farahmandi, Alif Ahmed, and Prabhat Mishra. 2018. Hardware Trojan detection using ATPG and model checking. In *2018 31st international conference on VLSI design and 2018 17th international conference on embedded systems (VLSID)*. IEEE, IEEE, 91–96.

[28] Thanh Nga Dang, Abhik Roychoudhury, Tulika Mitra, and Prabhat Mishra. 2009. Generating test programs to cover pipeline interactions. In *2009 46th ACM/IEEE Design Automation Conference*. IEEE, IEEE, 142–147.

[29] Shujun Deng, Kwang-Ting Cheng, Jinian Bian, and Zhiqiu Kong. 2010. Mutation-based diagnostic test generation for hardware design error diagnosis. In *2010 IEEE International Test Conference*, Vol. 1. IEEE Computer Society, 1–1.

[30] Andrew DeOrio, Daya Shanker Khudia, and Valeria Bertacco. 2011. Post-silicon bug diagnosis with inconsistent executions. In *2011 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 755–761.

[31] Paul E Dodd and Lloyd W Massengill. 2003. Basic mechanisms and modeling of single-event upset in digital microelectronics. *IEEE Transactions on nuclear Science* 50, 3 (2003), 583–602.

[32] Jack Edmonds and Ellis L Johnson. 1973. Matching, Euler tours and the Chinese postman. *Mathematical programming* 5 (1973), 88–124.

[33] Bernhard Egger, Eunjin Song, Hochan Lee, and Daeyong Shin. 2019. Random test program generation for verification and validation of the Samsung Reconfigurable Processor. *Journal of Systems Architecture* 97 (2019), 219–238.

[34] Marco Elver and Vijay Nagarajan. 2016. McVerSi: A test generation framework for fast memory consistency verification in simulation. In *2016 IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 618–630.

[35] Nestoras E Evmorfopoulos, Georgios I Stamoulis, and John N Avaritsiotis. 2002. A Monte Carlo approach for maximum power estimation based on extreme value theory. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 21, 4 (2002), 415–432.

[36] Farimah Farahmandi and Prabhat Mishra. 2016. Automated test generation for debugging arithmetic circuits. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, IEEE, 1351–1356.

[37] Farimah Farahmandi, Prabhat Mishra, and Sandip Ray. 2016. Exploiting transaction level models for observability-aware post-silicon test generation. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE).* IEEE, IEEE, 1477–1480.

[38] Harry Foster. 2020. 2020 Wilson research group functional verification study: IC/ASIC functional verification trend report. *Wilson Research Group and Mentor, A Siemens Business, White Paper* (2020).

[39] Nikos Foutris, Dimitris Gizopoulos, Mihalis Psarakis, Xavier Vera, and Antonio Gonzalez. 2011. Accelerating microprocessor silicon validation by exposing ISA diversity. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture.* 386–397.

[40] Ana T Freitas, Horácio C Neto, and Arlindo L Oliveira. 2000. On the complexity of power estimation problems. In *International Workshop on Logic Synthesis (ILWS).* 239–244.

[41] Franco Fummi. 2003. The confluence of manufacturing test and design validation. In *International Test Conference, 2003. Proceedings. ITC 2003.* IEEE Computer Society, 1291–1291.

[42] Daniel Geist, Monica Farkas, Avner Landver, Yossi Lichtenstein, Shmuel Ur, and Yaron Wolfsthal. 1996. Coverage-directed test generation using symbolic techniques. In *International Conference on Formal Methods in Computer-Aided Design.* Springer, 143–158.

[43] Frank Ghenassia et al. 2005. *Transaction-level modeling with SystemC.* Vol. 2. Springer.

[44] Google Fuzz Tester [n.d.]. Google Fuzz Tester. https://google.github.io/oss-fuzz/

[45] Tomás Grimm, Djones Lettnin, and Michael Hübner. 2018. A survey on formal verification techniques for safety-critical systems-on-chip. *Electronics* 7, 6 (2018), 81.

[46] Aarti Gupta. 1992. Formal hardware verification methods: A survey. In *Computer-Aided Verification.* Springer, 5–92.

[47] Onur Guzey and Li-C Wang. 2007. Coverage-directed test generation through automatic constraint extraction. In *2007 IEEE International High Level Design Validation and Test Workshop.* IEEE, IEEE, 151–158.

[48] Hadi Hajimiri, Kamran Rahmani, and Prabhat Mishra. 2015. Efficient peak power estimation using probabilistic cost-benefit analysis. In *2015 28th International Conference on VLSI Design.* IEEE, 369–374.

[49] Ian G Harris. 2003. The confluence of manufacturing test and design validation. In *International Test Conference, 2003. Proceedings. ITC 2003.* IEEE Computer Society, 1290–1290.

[50] Miao He, Jungmin Park, Adib Nahiyan, Apostol Vassilev, Yier Jin, and Mark Tehranipoor. 2019. RTL-PSC: Automated power side-channel leakage assessment at register-transfer level. In *IEEE VLSI Test Symposium (VTS).* 1–6.

[51] Ted Hong, Yanjing Li, Sung-Boem Park, Diana Mui, David Lin, Ziyad Abdel Kaleq, Nagib Hakim, Helia Naeimi, Donald S Gardner, and Subhasish Mitra. 2010. QED: Quick error detection tests for effective post-silicon validation. In *2010 IEEE International Test Conference.* IEEE, IEEE, 1–10.

[52] Yuanwen Huang, Swarup Bhunia, and Prabhat Mishra. 2016. MERS: statistical test generation for side-channel analysis based Trojan detection. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* 130–141.

[53] Jaewon Hur, Suhwan Song, Dongup Kwon, Eunjin Baek, Jangwoo Kim, and Byoungyoung Lee. 2021. DIFUZZRTL: Differential Fuzz Testing to Find CPU Bugs. In *2021 IEEE Symposium on Security and Privacy (SP).* IEEE, 1286–1303.

[54] Charalambos Ioannides and Kerstin I Eder. 2012. Coverage-directed test generation automated by machine learning–a review. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 17, 1 (2012), 1–21.

[55] Aruna Jayasena, Emma Andrews, and Prabhat Mishra. 2023. TVLA*: Test Vector Leakage Assessment on Hardware Implementations of Asymmetric Cryptography Algorithms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2023).

[56] Aruna Jayasena, Binod Kumar, Subodha Charles, Hasini Witharana, and Prabhat Mishra. 2022. Network-on-Chip Trust Validation using Security Assertions. *Journal of Hardware and Systems Security* (2022).

[57] Aruna Jayasena and Prabhat Mishra. 2023. HIVE: Scalable Hardware-Firmware Co-Verification using Scenario-based Decomposition and Automated Hint Extraction. *arXiv preprint arXiv:2309.08002* (2023).

[58] Aruna Jayasena and Prabhat Mishra. 2023. Scalable Detection of Hardware Trojans using ATPG-based Activation of Rare Events. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2023).

[59] Seonghun Jeong, Youngchul Cho, Daeyong Shin, Changyeon Jo, Yenjo Han, Soojung Ryu, Jeongwook Kim, and Bernhard Egger. 2012. Random test program generation for reconfigurable architectures. In *13th International Workshop on Microprocessor Test and Verification (MTV).*

[60] Christoph Kern and Mark R Greenstreet. 1999. Formal verification in hardware design: a survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 4, 2 (1999), 123–193.

[61] Nathan Kitchen and Andreas Kuehlmann. 2007. Stimulus generation for constrained random simulation. In *2007 IEEE/ACM International Conference on Computer-Aided Design*. IEEE, IEEE, 258–265.

[62] Tom Kolan, Hillel Mendelson, Vitali Sokhin, Shai Doron, Hernan Theiler, Shay Aviv, Hagai Hadad, Natalia Freidman, Elena Tsanko, John Ludden, et al. 2021. Post Silicon Validation of the MMU. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 212–217.

[63] Heon-Mo Koo and Prabhat Mishra. 2006. Functional test generation using property decompositions for validation of pipelined processors. In *Proceedings of the Design Automation & Test in Europe Conference*, Vol. 1. IEEE, IEEE, 1–6.

[64] Heon-Mo Koo and Prabhat Mishra. 2006. Test generation using SAT-based bounded model checking for validation of pipelined processors. In *Proceedings of the 16th ACM Great Lakes symposium on VLSI*. 362–365.

[65] Heon-Mo Koo and Prabhat Mishra. 2008. Specification-based compaction of directed tests for functional validation of pipelined processors. In *Proceedings of the 6th IEEE/ACM/IFIP international conference on Hardware/Software codesign and system synthesis*. 137–142.

[66] Heon-Mo Koo, Prabhat Mishra, Jayanta Bhadra, and Magdy Abadir. 2006. Directed micro-architectural test generation for an industrial processor: A case study. In *Seventh International Workshop on Microprocessor Test and Verification (MTV'06)*. IEEE, IEEE, 33–36.

[67] Harish Kriplani, Farid Najm, and Ibrahim Hajj. 1992. Maximum current estimation in CMOS circuits. In *[1992] Proceedings 29th ACM/IEEE Design Automation Conference*. IEEE, 2–7.

[68] Arun Krishnamachary, Jacob A Abraham, and Raghuram S Tupuri. 2001. Timing verification and delay test generation for hierarchical designs. In *VLSI Design 2001. Fourteenth International Conference on VLSI Design*. IEEE, IEEE, 157–162.

[69] Thomas Kropf and Hans-Joachim Wunderlich. 1991. A common approach to test generation and hardware verification based on temporal logic. In *1991, Proceedings. International Test Conference*. IEEE, IEEE, 57–66.

[70] Kevin Laeufer, Jack Koenig, Donggyu Kim, Jonathan Bachrach, and Koushik Sen. 2018. RFUZZ: Coverage-directed fuzz testing of RTL on FPGAs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–8.

[71] Nicole Lesperance, Shrikant Kulkarni, and Kwang-Ting Cheng. 2015. Hardware Trojan detection using exhaustive testing of k-bit subspaces. In *The 20th Asia and South Pacific Design Automation Conference*. IEEE, IEEE, 755–760.

[72] Rongjian Liang, Nathaniel Pinckney, Yuji Chai, Haoxin Ren, and Brucek Khailany. 2023. Late Breaking Results: Test Selection For RTL Coverage By Unsupervised Learning From Fast Functional Simulation. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–2.

[73] David Lin, Ted Hong, Farzan Fallah, Nagib Hakim, and Subhasish Mitra. 2012. Quick detection of difficult bugs for effective post-silicon validation. In *DAC Design Automation Conference 2012*. IEEE, IEEE, 561–566.

[74] Lingyi Liu and Shabha Vasudevan. 2009. STAR: Generating input vectors for design validation by static analysis of RTL. In *2009 IEEE International High Level Design Validation and Test Workshop*. IEEE, IEEE, 32–37.

[75] Lingyi Liu and Shobha Vasudevan. 2011. Efficient validation input generation in RTL by hybridized source code analysis. In *2011 Design, Automation & Test in Europe*. IEEE, IEEE, 1–6.

[76] Lingyi Liu and Shobha Vasudevan. 2014. Scaling input stimulus generation through hybrid static and dynamic analysis of RTL. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 20, 1 (2014), 1–33.

[77] Jason Lowe-Power. 2023. gem5: simulator system. Retrieved Octomber 10, 2023 from https://www.gem5.org

[78] David J. Lu. 1982. Watchdog processors and structural integrity checking. *IEEE Trans. Comput.* 31, 07 (1982), 681–685.

[79] Yangdi Lyu, Alif Ahmed, and Prabhat Mishra. 2019. Automated activation of multiple targets in RTL models using concolic testing. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, IEEE, 354–359.

[80] Yangdi Lyu and Prabhat Mishra. 2019. Efficient test generation for Trojan detection using side channel analysis. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, IEEE, 408–413.

[81] Yangdi Lyu and Prabhat Mishra. 2020. Automated test generation for Trojan detection using delay-based side channel analysis. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, IEEE, 1031–1036.

[82] Yangdi Lyu and Prabhat Mishra. 2020. Scalable activation of rare triggers in hardware trojans by repeated maximal clique sampling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 7 (2020), 1287–1300.

[83] Yangdi Lyu and Prabhat Mishra. 2021. MaxSense: Side-channel Sensitivity Maximization for Trojan Detection Using Statistical Test Patterns. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 26, 3 (2021), 1–21.

[84] Yangdi Lyu, Xiaoke Qin, Mingsong Chen, and Prabhat Mishra. 2018. Directed test generation for validation of cache coherence protocols. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38, 1 (2018), 163–176.

[85] Aamer Mahmood and Edward J McCluskey. 1988. Concurrent error detection using watchdog processors-a survey. *IEEE Trans. Comput.* 37, 2 (1988), 160–174.

[86] Microsoft Fuzz Tester [n.d.]. Microsoft Fuzz Tester. https://github.com/microsoft/onefuzz

[87] Prabhat Mishra, Swarup Bhunia, and Mark Tehranipoor. 2017. *Hardware IP security and trust*. Springer.

[88] Prabhat Mishra and Mingsong Chen. 2009. Efficient techniques for directed test generation using incremental satisfiability. In *2009 22nd International Conference on VLSI Design*. IEEE, IEEE, 65–70.

[89] Prabhat Mishra and Nikil Dutt. 2008. Specification-driven directed test generation for validation of pipelined processors. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 13, 3 (2008), 1–36.

[90] Prabhat Mishra and Farimah Farahmandi. 2019. *Post-Silicon Validation and Debug*. Springer.

[91] Prabhat Mishra, Ronny Morad, Avi Ziv, and Sandip Ray. 2017. Post-silicon validation in the SoC era: A tutorial introduction. *IEEE Design & Test* 34, 3 (2017), 68–92.

[92] Matthew W Moskewicz, Conor F Madigan, Ying Zhao, Lintao Zhang, and Sharad Malik. 2001. Chaff: Engineering an efficient SAT solver. In *Proceedings of the 38th annual Design Automation Conference*. 530–535.

[93] Rajdeep Mukherjee, Daniel Kroening, and Tom Melham. 2015. Hardware verification using software analyzers. In *2015 IEEE Computer Society Annual Symposium on VLSI*. IEEE, IEEE, 7–12.

[94] K Najeeb, Karthik Gururaj, V Kamakoti, and Vivekanand M Vedula. 2007. Controllability-driven power virus generation for digital circuits. In *20th International Conference on VLSI Design held jointly with 6th International Conference on Embedded Systems (VLSID'07)*. IEEE, 407–412.

[95] Seetharam Narasimhan, Dongdong Du, Rajat Subhra Chakraborty, Somnath Paul, Francis G Wolff, Christos A Papachristou, Kaushik Roy, and Swarup Bhunia. 2012. Hardware Trojan detection by multiple-parameter side-channel analysis. *IEEE Transactions on computers* 62, 11 (2012), 2183–2195.

[96] Seetharam Narasimhan, Xinmu Wang, Dongdong Du, Rajat Subhra Chakraborty, and Swarup Bhunia. 2011. TeSR: A robust temporal self-referencing approach for hardware Trojan detection. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, IEEE, 71–74.

[97] Yehuda Naveh, Michal Rimon, Itai Jaeger, Yoav Katz, Michael Vinov, Eitan s Marcu, and Gil Shurek. 2007. Constraint-based random stimuli generation for hardware verification. *AI magazine* 28, 3 (2007), 13–13.

[98] Chris Nigh and Alex Orailoglu. 2021. Adatrust: Combinational hardware trojan detection through adaptive test pattern construction. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 29, 3 (2021), 544–557.

[99] Yann Oddos, Katell Morin-Allory, Dominique Borrione, Marc Boulé, and Zeljko Zilic. 2009. Mygen: Automata-based on-line test generator for assertion-based verification. In *Proceedings of the 19th ACM Great Lakes symposium on VLSI*. 75–80.

[100] Nahmsuk Oh, Philip P Shirvani, and Edward J McCluskey. 2002. Error detection by duplicated instructions in super-scalar processors. *IEEE Transactions on Reliability* 51, 1 (2002), 63–75.

[101] Zhixin Pan and Prabhat Mishra. 2021. Automated test generation for hardware trojan detection using reinforcement learning. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*. 408–413.

[102] Zhixin Pan, Jennifer Sheldon, and Prabhat Mishra. 2020. Test generation using reinforcement learning for delay-based side-channel analysis. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–7.

[103] George Papadimitriou, Athanasios Chatzidimitriou, Dimitris Gizopoulos, and Ronny Morad. 2016. An Agile Post-Silicon Validation Methodology for the Address Translation Mechanisms of Modern Microprocessors. *IEEE Transactions on Device and Materials Reliability* 17, 1 (2016), 3–11.

[104] George Papadimitriou, Dimitris Gizopoulos, Athanasios Chatzidimitriou, Tom Kolan, Anatoly Koyfman, Ronny Morad, and Vitali Sokhin. 2016. Unveiling difficult bugs in address translation caching arrays for effective post-silicon validation. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*. IEEE, 544–551.

[105] Irith Pomeranz and Sudhakar M Reddy. 2004. A measure of quality for n-detection test sets. *IEEE transactions on computers* 53, 11 (2004), 1497–1503.

[106] Nitin Pundir, Jungmin Park, Farimah Farahmandi, and Mark Tehranipoor. 2022. Power Side-Channel Leakage Assessment Framework at Register-Transfer Level. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* (2022).

[107] Xiaoke Qin, Mingsong Chen, and Prabhat Mishra. 2010. Synchronized generation of directed tests using satisfiability solving. In *2010 23rd International Conference on VLSI Design*. IEEE, IEEE, 351–356.

[108] Xiaoke Qin and Prabhat Mishra. 2012. Automated generation of directed tests for transition coverage in cache coherence protocols. In *2012 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, IEEE, 3–8.

[109] Xiaoke Qin and Prabhat Mishra. 2012. Directed test generation for validation of multicore architectures. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 17, 3 (2012), 1–21.

[110] Xiaoke Qin and Prabhat Mishra. 2014. Scalable test generation by interleaving concrete and symbolic execution. In *2014 27th International Conference on VLSI Design and 2014 13th International Conference on Embedded Systems*. IEEE, IEEE, 104–109.

[111] Xiaoke Qin, Weixun Wang, and Prabhat Mishra. 2012. TCEC: Temperature and energy-constrained scheduling in real-time multitasking systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31, 8 (2012), 1159–1168.

[112] Danilo Ravotto, Ernesto Sánchez, Massimiliano Schillaci, and Giovanni Squillero. 2008. An evolutionary methodology for test generation for peripheral cores via dynamic FSM extraction. In *Workshops on Applications of Evolutionary Computation*. Springer, Springer, 214–223.

[113] Mohammad Sabri, Ahmad Shabani, and Bijan Alizadeh. 2021. SAT-Based Integrated Hardware Trojan Detection and Localization Approach Through Path-Delay Analysis. *IEEE Transactions on Circuits and Systems II: Express Briefs* 68, 8 (2021), 2850–2854.

[114] Sayandeep Saha, Rajat Subhra Chakraborty, Srinivasa Shashank Nuthakki, Debdeep Mukhopadhyay, et al. 2015. Improved test pattern generation for hardware trojan detection using genetic algorithm and boolean satisfiability. In *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, Springer, 577–596.

[115] Jacob Savir. 1997. Delay test generation: a hardware perspective. *Journal of Electronic Testing* 10, 3 (1997), 245–254.

[116] Yiannakis Sazeides, Alex Gerber, Ron Gabor, Arkady Bramnik, George Papadimitriou, Dimitris Gizopoulos, Chrysostomos Nicopoulos, Giorgos Dimitrakopoulos, and Karyofyllis Patsidis. 2022. Idld: Instantaneous detection of leakage and duplication of identifiers used for register renaming. In *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 799–814.

[117] Divjyot Sethi, Muralidhar Talupur, and Sharad Malik. 2014. Using flow specifications of parameterized cache coherence protocols for verifying deadlock freedom. In *Automated Technology for Verification and Analysis*. Springer, 330–347.

[118] Zhendong Shi, Haocheng Ma, Qizhi Zhang, Yanjiang Liu, Yiqiang Zhao, and Jiaji He. 2021. Test Generation for Hardware Trojan Detection Using Correlation Analysis and Genetic Algorithm. *ACM Transactions on Embedded Computing Systems (TECS)* 20, 4 (2021), 1–20.

[119] Douglas J Smith. 1996. VHDL and Verilog compared and contrasted-plus modeled example written in VHDL, Verilog and C. In *33rd Design Automation Conference Proceedings, 1996*. IEEE, IEEE, 771–776.

[120] Paul Stephan, Robert K Brayton, and Alberto L Sangiovanni-Vincentelli. 1996. Combinational test generation using satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15, 9 (1996), 1167–1176.

[121] Donald Thomas and Philip Moorby. 2008. *The Verilog® hardware description language*. Springer Science & Business Media.

[122] Jason G Tong, Marc Boulé, and Zeljko Zilic. 2013. Test compaction techniques for assertion-based test generation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 19, 1 (2013), 1–29.

[123] Shobha Vasudevan, Wenjie Joe Jiang, David Bieber, Rishabh Singh, C Richard Ho, Charles Sutton, et al. 2021. Learning semantic representations to verify hardware designs. *Advances in Neural Information Processing Systems* 34 (2021), 23491–23504.

[124] Freek Verbeek, Pooria M Yaghini, Ashkan Eghbal, and Nader Bagherzadeh. 2016. Deadlock verification of cache coherence protocols and communication fabrics. *IEEE Trans. Comput.* 66, 2 (2016), 272–284.

[125] Manisha Vinta and S Sivanantham. 2020. Modeling and Test Generation for Combinational Hardware Trojans. In *2020 IEEE International Test Conference India*. IEEE, IEEE, 1–4.

[126] Ilya Wagner and Valeria Bertacco. 2008. MCjammer: Adaptive verification for multi-core designs. In *Proceedings of the conference on design, automation and test in Europe*. 670–675.

[127] Ilya Wagner and Valeria Bertacco. 2008. Reversi: Post-silicon validation system for modern microprocessors. In *2008 IEEE International Conference on Computer Design*. IEEE, 307–314.

[128] Chuan-Yu Wang and Kaushik Roy. 1996. Maximum power estimation for CMOS circuits using deterministic and statistic approaches. In *Proceedings of 9th International conference on VLSI design*. IEEE, 364–369.

[129] Chuan-Yu Wang and Kaushik Roy. 1997. Estimation of maximum power for sequential circuits considering spurious transitions. In *Proceedings International Conference on Computer Design VLSI in Computers and Processors*. IEEE, 746–751.

[130] Weixun Wang, Xiaoke Qin, and Prabhat Mishra. 2010. Temperature-and energy-constrained scheduling in multitasking systems: A model checking approach. In *2010 ACM/IEEE International Symposium on Low-Power Electronics and Design (ISLPED)*. IEEE, 85–90.

[131] Hasini Witharana, Yangdi Lyu, Subodha Charles, and Prabhat Mishra. 2022. A Survey on Assertion-based Hardware Verification. *ACM Computing Surveys (CSUR)* 54, 11s (2022), 1–33.

[132] Hasini Witharana, Yangdi Lyu, and Prabhat Mishra. 2021. Directed test generation for activation of security assertions in rtl models. *ACM Transactions on Design Automation of Electronic Systems (TODAES)* 26, 4 (2021), 1–28.

[133] David A Wood, Garth A Gibson, and Randy H Katz. 1990. Verifying a multiprocessor cache controller using random test generation. *IEEE Design & Test of Computers* 7, 4 (1990), 13–25.

[134] Meng Zhang, Jesse D Bingham, John Erickson, and Daniel J Sorin. 2014. PVCoherence: Designing flat coherence protocols for scalable verification. In *International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 392–403.

[135] Tao Zhang, Jungmin Park, Mark Tehranipoor, and Farimah Farahmandi. 2021. PSC-TG: RTL power side-channel leakage assessment with test pattern generation. In *ACM/IEEE Design Automation Conference (DAC)*. 709–714.