Functional Coverage Closure in SoC Interconnect Verification with Iterative Machine Learning

Jihye Kwon*, Sukwon Ha†, Youngsik Kim†, Seonil Brian Choi†, Daeseo Cha†, Space Kim†, Kunhyuk Kang†, John Pierce*, Amit Metodi*, Saurabh Sharma*, Heedo Jung*, Yosinori Watanabe*

† Samsung Electronics Co., Ltd., Seoul, Korea
{sukwon.ha, ys31.kim, seonilb.choi, dscha, space.kim, kunhyuk.kang}@samsung.com

* Cadence Design Systems, {US, Israel, India, Korea}
{kwonj, jlp, ametodi, shsaurab, hdjung, watanabe}@cadence.com

***Abstract*-System-on-Chip (SoC) interconnects connect various components of the SoC. The ever-increasing SoC design complexity has brought about boosted complexity in SoC interconnect designs that connect hundreds of master and slave components with various protocols. Functional verification of such SoC interconnect designs constitutes a critical part of the SoC design process. To reach the functional coverage closure, the design verification team expends excessive time and resources on running constrained random simulations, analyzing uncovered scenarios, and modifying the tests and settings in turn. This paper presents a novel and alternative approach that exploits iterative machine learning by adopting a software solution called SimAI. Given the original set of tests and a lengthy list of target scenarios, the software generates instructions for Xcelium Logic Simulator to address those target scenarios efficiently. Moreover, by evaluating simulation traces and results, it generates a new set of instructions to address the remaining targets in subsequent iterations. Experimental results on a production project demonstrates that this approach can greatly reduce the total simulation time and human effort for functional coverage closure, up to 18x reduction in CPU hours to increase 20 %pt coverage to eventually achieve the target coverage closure.**

I. Introduction

As the scale and complexity of System-on-Chip (SoC) designs continue to increase, the verification challenge for those designs has also been growing exponentially. In particular, SoC interconnects form the backbone of SoC designs, making it crucial to thoroughly verify them at early stages of design. However, the complete verification of an SoC interconnect poses a significant challenge due to the connection of hundreds of master and slave intellectual property (IP) cores with different protocols and the routing of various types of transactions [1]. State-of-the-art verification approaches involve a scalable Universal Verification Methodology (UVM) testbench that reflects the interconnect topology, constrained random simulation techniques that generate massive random transactions, and a huge functional coverage model that ensures a thorough verification of the design under test (DUT) [2], [3]. Consequently, it takes the design verification team an enormous amount of time and resources to achieve functional coverage closure by repeatedly running simulations after manually changing constraints in the testbench to target uncovered areas.

This paper proposes a novel methodology to effectively reduce the time and cost for functional coverage closure in SoC interconnect verification by iteratively applying artificial intelligence and machine learning solutions that have been packaged as a feature of SimAI from Cadence Design Systems [4]. To start with, the design verification team creates a set of tests as usual and specifies target scenarios. The verification team initially simulates the set of tests using Xcelium Logic Simulator [5]. Then, SimAI analyzes the simulation traces and results in order to generate instructions that guide the simulator to reach the target scenarios efficiently. After the verification team simulates the tests with the generated instructions, SimAI evaluates those simulation traces and results to generate a new set of instructions for addressing the remaining targets. By running this flow iteratively, the verification team can step toward closing the target coverage without manually modifying the tests or simulation settings.

The contributions of this paper can be summarized as follows: (i) It adopts iterative machine learning to efficiently address a multitude of target scenarios for SoC interconnect verification that are extremely difficult and demanding to address by simulating the original tests repeatedly; (ii) We applied the proposed approach to a production project to achieve 100% coverage for two challenging sets of coverage bins, spending 4.92x and 18.20x less CPU time than repeatedly simulating the original set of tests, which could only reach 96.27% and 79.91% coverage, respectively.

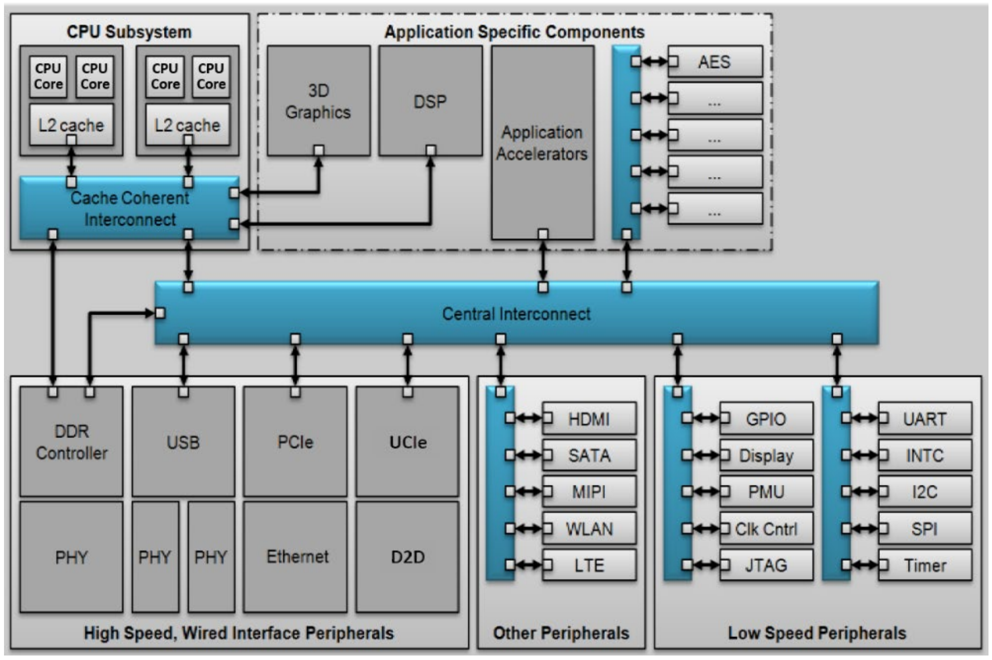

Figure 1. A typical SoC with cascaded interconnects.

II. Background and Challenges

An SoC interconnect is the communication backbone within an SoC that connects multiple subsystems such as processors (e.g., CPU, GPU, and application-specific processors), memory, I/O interface, and peripheral devices. It plays a crucial role in managing data flows, optimizing performance, ensuring security, and maintaining low power consumption of modern SoCs. The design complexity of SoC interconnects has increased due to (i) the integration of numerous IP components with different bus interface protocols, and (ii) the requirements for high performance and power efficiency. To meet these demands, today's SoC interconnects typically consist of a cascade of interconnects including the Cache Coherent Interconnect, Network-on-Chip (NoC), AXI/AHB interconnect, and multiple bridges, as shown in Figure 1 [1].

Due to the increasing number of IP components and various bus interface protocols, scalability has arisen as a critical challenge to the verification of SoC interconnects. In a single SoC interconnect, hundreds of master and slave components establish hundreds of thousands of data paths, each of which may transfer data with various attributes such as address, direction, transfer size, and other control or data values defined in the bus interface protocols. The data paths, types, and attributes to be tested are captured in the functional coverage model. The number of target coverage bins (including coverpoint bins and cross-coverage bins) has grown in scale with ever-increasing numbers of masters and slaves.

Functional verification methodology for industrial designs has centered around constrained random simulation [6]. For an SoC interconnect, constrained random tests generate random transactions between a master and slaves in order to validate that data are correctly routed between the master and the designated slaves according to the design specification [7]. By simulating the same tests numerous times, the design verification team can simulate a huge volume of random traffic originated from hundreds of masters toward hundreds of slaves, at the cost of the increased total simulation time due to the increased complexity.

Yet, it still remains challenging to reach the functional coverage closure by repeatedly simulating those constrained random tests. In order to reach a state in the simulation where a target coverage bin can be hit, a certain combination of randomizations should have happened toward that state. The required simulation settings and randomization combinations are distinct for each coverage bin. In general, a random test possesses opportunities to reach diverse states of the simulation; however, it is tricky to reach every state in an extensive list of states that are associated with target coverage bins. On the other hand, a directed test can be written to target a specific state; nevertheless, given the huge coverage model and verification complexity, it is infeasible to generate and simulate a directed test for every coverage bin.

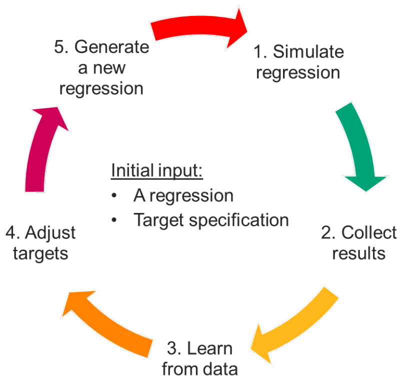

Figure 2. An overview of the proposed flow.

Hence, the design verification team usually simulates the constrained random tests repeatedly to hit as many bins as possible, and then modifies the testbench or simulation settings to directly target the unhit bins. Rarely, they confirm that a few unhit bins are indeed unreachable and exclude those bins from the coverage model after a thorough analysis. As both the coverage model size and the simulation time have increased for SoC interconnects, it has become extremely difficult to close the coverage and meet the project schedule with the conventional verification approach.

III. Proposed Approach

Large-scale verification projects typically involve periodically simulating a suite of various constrained random tests, called a regression. Instead of simulating the same regression repeatedly or modifying the testbench to directly test a few specific cases as in the conventional approach, the proposed approach is to leverage a software solution called SimAI to iteratively and automatically generate a new regression based on the simulation results of the previous regression. Here, a newly generated regression consists of the original regression itself and a separate list of instructions for Xcelium Logic Simulator. While SimAI offers multiple objective functions in generating regressions such as autonomous coverage maximization or bug hunting, the proposed approach uses the one that generates regressions to realize user-specified test scenarios efficiently.

Figure 2 presents a high-level overview of the proposed iterative flow. Initially, the design verification team develops a regression and produces a target specification. The regression mainly consists of a number of constrained random tests and their random seed counts. The target specification contains a list of target scenarios. For SoC interconnect verification, scenarios can be defined as simulation settings of individual tests and combinations of values of random variables in the testbench that correspond to the routing paths and transaction attributes to be tested, according to the coverage model. That is, if particular scenarios occur in simulation, corresponding coverage bins will be hit. This information may be extracted from the coverage model or results, and the testbench or simulation traces.

In Step 1, the verification team simulates the original regression using Xcelium Logic Simulator. Each random test with a single random seed becomes a single simulation run. In addition to simulating the tests, Xcelium also produces simulation data for SimAI: simulation traces, status information, and coverage results. The trace of a simulation run includes information about which random variables were randomized to which values at which randomize calls. The status of a run refers to whether the simulation successfully completed or presented an error. For a failed run, the status information includes the error or failure message. The coverage results of a run include the functional and code coverage results.

After simulation, the verification team launches SimAI, providing the simulation data from the original regression and the target scenario specification. Then, a feature of SimAI performs Steps 2 – 5 and generates a new regression to be simulated. In Step 2, it collects simulation data of the input regression and stores them after pre-processing.

In Step 3, SimAI learns from the data collected in Step 2. By analyzing the simulation traces, it identifies all target scenarios in the input specification that have been *satisfied* by simulation runs in the input regression. A scenario is satisfied by a simulation run if the combination of randomizations of each variable to the value range specified in the scenario occurs at least once in the specified simulation setting during the run. SimAI also evaluates the difficulty of satisfying the remaining target scenarios by analyzing the collected data in terms of which random variables appeared in the simulation runs of the regression and how they were randomized. For instance, it is more difficult to satisfy a scenario that specifies a random variable to be randomized to a value that has not been observed in simulation traces.

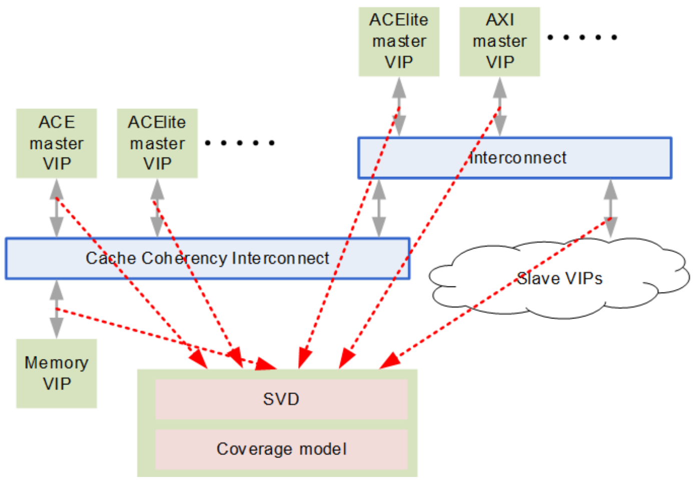

Figure 3. The SoC interconnect testbench.

In Step 4, SimAI adjusts targets according to the evaluation results from Step 3. It decides which target scenarios should be addressed in a new regression. It also decides which of them will be jointly included in individual simulation runs of the new regression by analyzing their relationship such as contradiction and independence. Two target scenarios are said to be contradictory if both require that a common random variable should be randomized at the same time but to disjoint value ranges. Each of the contradictory scenarios requires a separate simulation run to address it. On the other hand, a pair of target scenarios may be satisfied by randomizing distinct random variables. Such target scenarios are said to be independent, and they become a candidate to be targeted in a single simulation run. SimAI utilizes such relationship among target scenarios when constructing a new regression.

In Step 5, SimAI generates a new regression that aims to efficiently simulate the target scenarios selected in Step 4. The generated regression includes all or a subset of the initial regression's tests; to be specific, it includes a pointer to the initial regression files (e.g., the testbench code), and the number of random seeds applied for each test. In addition, for each simulation run, the generated regression contains sets of instructions that guide Xcelium Logic Simulator to steer randomizations so that specific, possibly multiple, target scenarios occur in the run as scheduled in Step 4.

Once a new regression is generated, the verification team simulates it, starting a new iteration from Step 1. They repeat this process iteratively until all target scenarios are addressed, at which point the functional coverage will be closed. Addressing a large number of target scenarios in a single simulation run may result in elongated simulation time; however, this overhead turned out to be minor compared to the total simulation time required to close the coverage using the conventional approach, as reported in the next section.

IV. Experimental Results

A. *Target design and testbench*

We applied the proposed approach to the SoC interconnect design for a state-of-the-art mobile application processor. The target interconnect consists of multi-layer interconnects such as the Cache Coherent Interconnect, NoC, AXI interconnect, AHB bus matrix, and multiple bridges. Hundreds of masters and near a thousand slaves are connected to the interconnects, resulting in hundreds of thousands of functional coverage elements.

The SoC interconnect verification environment is created in the early stages of SoC design by attaching active Verification IPs (VIPs) to the target interconnect in place of all master and slave components. (Instead of active VIPs that replace the components, passive VIPs can also be attached to monitor signals between the interconnect and the component DUTs.) Figure 3 illustrates an overview of the target testbench comprised of (i) master VIPs that generate random master sequences to the SoC interconnect, (ii) slave VIPs that react to the master sequences, (iii) System Verification Scoreboard (SVD) VIP that checks the end-to-end coherency, correctness, and integrity of data as they pass through the SoC interconnect [8], and (iv) the functional coverage model. The testbench has been generated by a UVM-based approach that facilitates the integration with VIPs and supports seamless configuration for the given target interconnect topology [9].

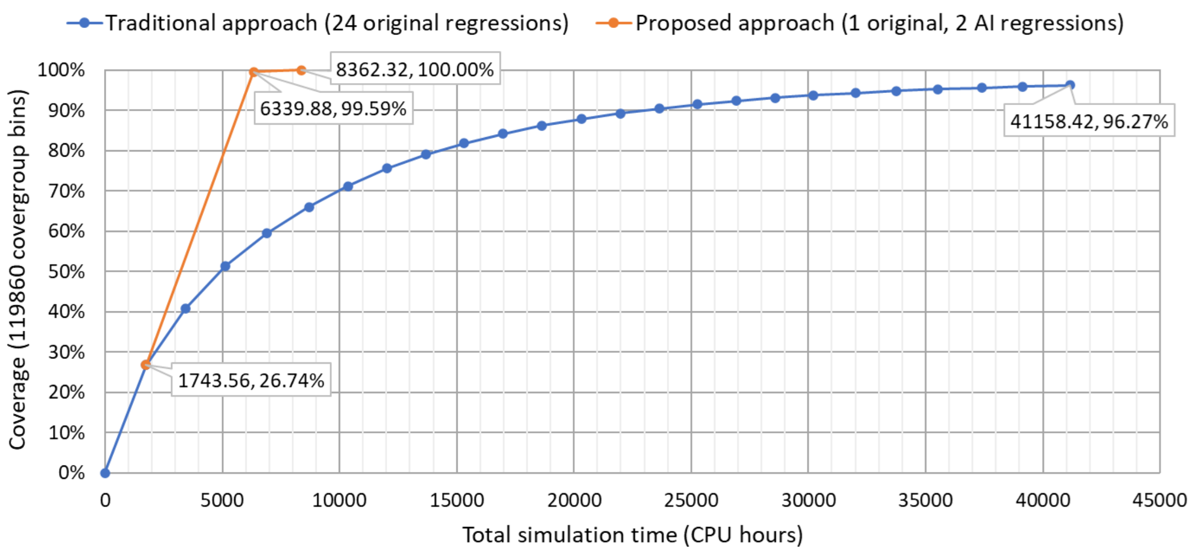

Figure 4. Coverage and total simulation time for Category 1.

From the functional coverage model, two categories of covergroup items have been identified as demanding a tremendous amount of human effort, time, and compute resources for the design verification team in the actual project to achieve the coverage closure.

B. Category 1

Cover items in the first category check the master and slave control signals such as the address, data, and direction, with respect to elements of the reachability matrix that indicates for each master the accessible slaves, inaccessible slaves, and unused addresses in the system. The coverage model contains over 500 such cover items, defining about 120,000 coverage bins in total.

Figure 4 shows the simulation results of the traditional and proposed approaches for Category 1. The traditional approach is to simulate the original regression repeatedly. The original regression contains 274 constrained random runs that consumed 1,744 CPU hours in total for the initial simulation to cover 26.74 % of the coverage bins in Category 1. In the figure, each point (a blue filled circle) marks the accumulated coverage and total simulation time measured at each new simulation of the regression. Simulating this regression 24 times resulted in 6,576 simulation runs, taking 41,158 CPU hours in total to cover 96.27 % of the bins. As one may note from the blue trend line in the figure, the higher the coverage is, the more simulation runs are required to further increase the coverage. This is due to the diminishing marginal return; the first 20 iterations (5,480 runs, 33,751 CPU hours) covered 94.84 % of the bins, and the additional four iterations (1,096 runs, 7,408 CPU hours) newly covered only 1.43 % of the bins.

The proposed approach takes the original regression and its initial simulation results to generate new regressions using SimAI. The first regression generated by SimAI contains 233 individual simulation runs that consumed about 4,600 CPU hours to hit additional 72.85 % of the bins. Although the average simulation time for a run has increased, those runs were guided to address many target scenarios efficiently. As a result, 99.59 % of the bins in Category 1 were hit after the total simulation time of 6,340 CPU hours. In the next iteration, SimAI generated a new regression containing 265 individual simulation runs, aiming to address the remaining target scenarios. It took 2,022 CPU hours to simulate this regression and hit all the uncovered bins. By simulating two AI regressions in addition to the initial original regression, the proposed approach spent 8,362 CPU hours (772 simulation runs) in total to cover 100 % of the target bins. This trajectory is marked in Figure 4 by the orange trend line that branches out from the point that represents the initial simulation result of the original regression. In summary, the proposed approach achieved both 4.92x reduction in CPU time for simulation and 3.73 %pt increase (from 96.27 % to 100.00 %) in coverage than the traditional approach.

Moreover, the proposed approach achieved the coverage closure for this category. In the traditional approach, after consuming 41,158 CPU hours for 24 iterations, the design verification team would start examining how the remaining 4,478 uncovered bins could be hit. As a common practice, in order to hit these bins, we often modify the tests and settings to directly simulate target scenarios, due to the diminishing marginal return of simulating the same regression repeatedly. This kind of examination of remaining coverage bins and possible modifications of tests require human

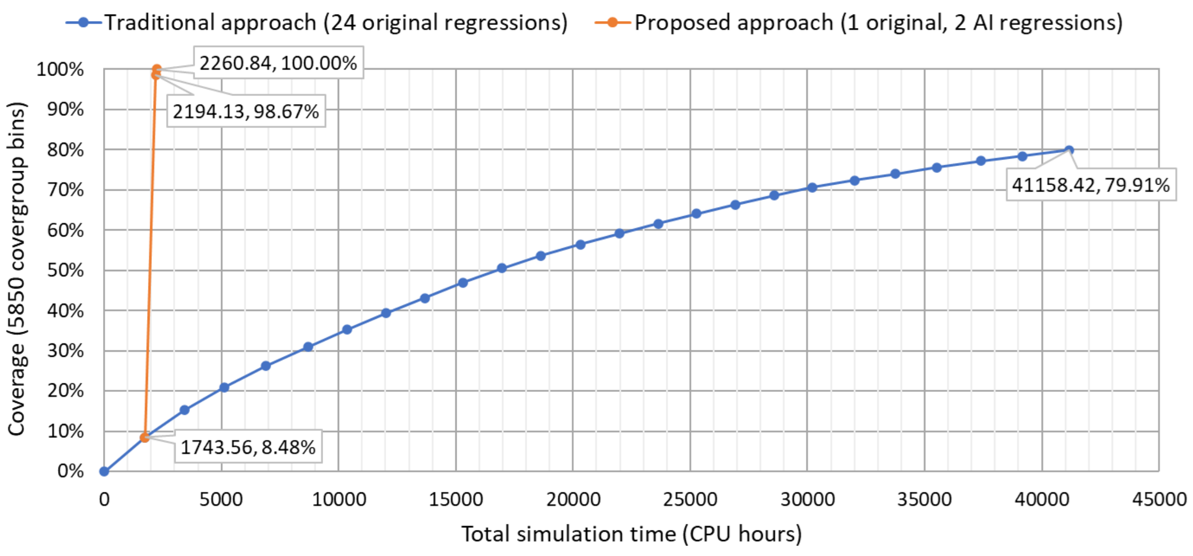

Figure 5. Coverage and total simulation time for Category 2.

engineering time. Hence, the total amount of time and resources saved by the proposed approach is more than just the reduction in simulation time of regressions. This approach also saves additional human effort required to close the coverage by the traditional approach.

C. Category 2

Cover items in the second category check the quality of service (QoS) value transmitted from the masters to the target points. The objective of these cover items is to validate the transactions for all valid QoS values. 496 cover items in this category contain 5,850 coverage bins in total.

Figure 5 shows the simulation results of the traditional and proposed approaches for Category 2. The traditional approach here is the same as that for Category 1 in Section B above, but the coverage results for Category 2 are presented by blue points in this figure. After the initial simulation of the original regression that contains 274 constrained random runs, 8.48 % of the coverage bins in Category 2 were hit at the cost of 1,744 CPU hours in total. Simulating the original regression 24 times resulted in hitting 79.91 % of the bins after spending 41,158 CPU hours (6,576 simulation runs) in total.

The proposed approach diverges from the traditional one after the initial simulation of the original regression. In the first iteration, SimAI generated a new regression with 48 individual simulation runs to hit additional 90.19 % of the coverage bins in Category 2 at the cost of 451 CPU hours for simulation. The second regression generated in the next iteration contains seven individual simulation runs that turned out to hit all the remaining bins of this category when simulated, spending 67 CPU hours. As marked in Figure 5 by the orange trend line that branches out from the initial simulation result of the original regression, the proposed approach closed the coverage gap of 91.52 % in two iterations at the additional cost of 517 CPU hours. Overall, this approach spent 2,261 CPU hours (329 simulation runs) in total to reach 100 % coverage, reducing the simulation time by 18.20x and at the same time increasing the coverage by 20.09 %pt (from 79.91 % to 100.00 %) than the traditional approach.

Furthermore, the proposed approach completely closed the coverage for this category as well. In the traditional approach, after simulating the original regression 24 times, the design verification team would either continue simulating it repeatedly to increase the coverage slowly or inspect the 1,175 uncovered bins and modify the regression to directly target each of those bins.

D. Summary

For the two challenging sets of coverage items in the production project, the proposed approach has achieved the coverage closure with 4.92x and 18.20x reduction in simulation time, respectively. On top of that, it does not require the enormous amount of human effort to examine and cover the remaining bins as often required in the traditional approach. Overall, the proposed approach has been demonstrated to significantly reduce time, compute resources, and human effort for functional coverage closure in SoC interconnect verification.

V. Concluding Remarks

It is becoming increasingly challenging to complete SoC interconnect verification in the early stages of design. We have proposed a novel approach that has the potential to accelerate the verification process by iteratively adjusting and addressing target scenarios with the aid of machine learning.

In the conventional approach, the design verification team repeatedly simulates a set of constrained random tests for many iterations and modifies the regression as needed in order to directly address uncovered bins. Instead of repeatedly simulating the same regression, the proposed approach leverages a feature of SimAI to generate a new regression that aims at simulating uncovered target scenarios efficiently based on the evaluation results of the previous simulation data and the relationship among the scenarios. By applying this approach to an industrial SoC interconnect verification project, we have demonstrated that it can significantly reduce the total simulation time as well as human effort to achieve the functional coverage closure, compared to the traditional approach.

We note that the effectiveness of the proposed approach depends on the completeness and accuracy of the target specification provided by the design verification team. While there is no preferred coding style for the coverage model, this approach assumes that a certain combination of randomizations in simulation traces will hit specific coverage bins being concerned. If the target specification is incomplete, i.e. it does not include test scenarios necessary to hit specific coverage bins, the proposed approach may not hit the bins even though all the test scenarios of the provided specification are satisfied. Future work includes deriving the target specification automatically.

Acknowledgment

The authors thank Alex Kondratyev, Michele Petracca, Jonathan Quijas, Andrea Rossolini, Shu-Ping Chen, Andrew Lynch, Paresh Varshney, Tal Zlotnick, and Yuval Konrad for developing algorithms, infrastructure, and simulation control mechanisms for SimAI. The authors also thank, Deepak Chauhan, Namit Sahu, Deepak Kumar, Vardhani Harpanahalli, Ben Allen, Vignesh Thiagarajan, David Eastham, Manu Bhatnagar, John Rose, Aanchal Sachdeva, and Anika Sunda for their continuous support on SimAI. The authors are grateful to Sang-Jun Mun and Lois Lee for their support on this collaborative work.

References

[1] A. B. Mehta, "SoC Interconnect Verification," in *ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies*, Springer International Publishing, 2018, pp. 273--284.

[2] "IEEE Standard for System, Software, and Hardware Verification and Validation," *IEEE Std 1012-2016,* pp. 1-260, 2017.

[3] "IEEE Standard for Universal Verification Methodology Language Reference Manual," *IEEE Std 1800.2-2020 (Revision of IEEE Std 1800.2-2017),* pp. 1-458, 2020.

[4] "Verisium AI-Driven Verification Platform," Cadence Design Systems, [Online]. Available: www.cadence.com/en_US/home/tools/system-design-and-verification/ai-driven-verification.html. [Accessed October 2024].

[5] "Xcelium Logic Simulator," Cadence Design Systems, [Online]. Available: www.cadence.com/en_US/home/tools/system-design-and-verification/simulation-and-testbench-verification/xcelium-simulator.html. [Accessed October 2024].

[6] N. Kitchen and A. Kuehlmann, "Stimulus Generation for Constrained Random Simulation," in *2007 IEEE/ACM International Conference on Computer-Aided Design*, 2007.

[7] K. S. Pooja, S. Krishnakumar and H. V. R. Aradhya, "Verification of Interconnection IP for Automobile Applications using System Verilog and UVM," in *2018 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2018.

[8] "System VIP Technical Brief," Cadence Design Systems, [Online]. Available: https://www.cadence.com/en_US/home/resources/technical-briefs/system-vip-tb.html. [Accessed October 2024].

[9] J. Bromley, "If SystemVerilog is so good, why do we need the UVM? Sharing responsibilities between libraries and the core language," in *Proceedings of the 2013 Forum on specification and Design Languages (FDL)*, 2013.