Accelerating Functional Verification Through Stabilization of Testbench Using AI/ML

Srikanth Vadanaparthi[1], Pooja Ganesh[1], Dharmesh Mahay[2], Malay Ganai[2]
[1] Qualcomm, 5775 Morehouse Dr, San Diego, CA 92121
[2] Synopsys, 690 East Middlefield Rd, Mountain View, CA 94043

***Abstract*-A Constrained Random Testbench plays a key role in functional verification but fine-tuning it is non-trivial, laborious, and risk prone. Some testbench issues can slow down the entire verification process and make it harder to achieve coverage closure in a timely manner. This paper proposes a methodology of adopting AI/ML early in the verification stages to stabilize (i.e., mature) the testbench faster. Using this methodology, we achieved approximately two weeks of savings, especially through a reduction in the manual effort to write directed tests, in the number of regressions to find corner case bugs, and a 10-15% reduction per block of grid resource usage.**

I. INTRODUCTION

A constrained random testbench plays a key role in verification sign-off, but fine-tuning the stimulus constraints is non-trivial, laborious, and risk-prone. Primary source of the testbench issues such as illegal stimuli, missing stimuli, and over and under bias, stems from the misalignment of the implemented constraint stimuli space and the intended stimuli space. These issues can lead to longer verification cycles, reduced verification efficiency, costly debugging, and even more severe issues like bug escapes due to omission scenarios [1]. Testbench developers can face various practical challenges such as no visibility to the constrained stimuli distribution, under- or over-biasing, under- or over- constraining, evolving coverage specifications, and debugging testbench failures. The current solutions (such as interactive debugging) are not most applicable for debugging or optimizing the test regression or exposing these testbench issues automatically. Given the resource and time limitations in the verification cycle of a chip, it can be challenging and time-consuming to mature testbenches that can target a diverse range of areas in the design space to find both testbench and design bugs, while trying to hit the coverage targets efficiently.

In projects, bugs found by the designers can be broadly categorized into the following three categories:

1. *Easy-to-find bugs:* Usually, these bugs can be exposed by a set of input or input sequences that systematically sweeps through all input combinations.
2. *Corner scenario or hard-to-hit bugs*: These bugs are typically found in the following:
    i) Complex logic with independent inputs
    ii) The logic which requires a long loop of action to trigger
    iii) Rare occurrence scenarios that may show up and disappear.
3. *Dead bugs*: These bugs are latent RTL bugs hidden so deep that the designer would never know.

Our block level Design Verification (DV) environments are built through careful consideration to hit Type #1 bugs during the initial phase of projects. The Type #2 bug category typically needs random regressions, good stimuli, and scenarios to hit them. These bugs are hit mostly during the late stage of project milestones, and in a few cases, these bugs escape to silicon, which must be avoided. Type #3 category bugs exist deep inside RTL and need months of regression to hit them. It depends on scenario generation quality, delay profile randomization, and input stimulus generation quality.

Graphics Processing Unit (GPU) architectures are designed to have great amount of parallelism to achieve high performance. This parallelism provides huge challenges on finding bugs in small corners of complex logic as well as closing coverage of both code and functional on time. A significant manual effort is needed to write semi-directed

test cases to hit corner cases at the block level and cluster level, further increasing the time taken to close functional and code coverage at the block level. Our team was exploring various functional verification solutions to shift-left the functional coverage closure. We posed the following criteria for adopting such solutions (preferably an AI/ML based):

a. Shift-left the finding of corner case RTL, testbench, and constraint issues
b. Accelerate functional or code coverage automatically
c. Help reduce the manual effort to write test cases of direct scenarios for hard-to-hit scenarios
d. No manual effort to rewrite or change functional coverage models

II. OUR APPROACH

With our goal to accelerate functional verification through the stabilization of the testbench, we wanted to share our experience of adopting an AI/ML technique such as Intelligent Coverage Optimization (ICO) in VCS® [2,3] early in our verification cycle. ICO technology is designed to improve the stimuli quality through diversification in the constraint stimuli space without requiring any rewrite of functional coverage models. We demonstrate how we leverage such AI/ML technology using multiple real-scenario case studies.

In a controlled experimentation, we observed an additional 30% rate of testbench issues manifested while using ICO in a testbench deemed to be stable (Figure 1). These additional issues comprised constraint inconsistency failures, design SVA failures due to issues in the testbench UVM driver, scoreboard checker issues, and some deadlock scenarios. Moreover, we achieved the same 95% functional coverage 40% faster with ICO, i.e., we met coverage with only 6 iterations of regression using ICO as opposed to 10 iterations without it.

We learned that fixing the testbench issues and optimizing the regression in the late stage of the project is comparatively expensive. A methodology using AI/ML that can learn and improve the stimuli on the fly by providing useful testbench analytics to the user may help achieve testbench stabilization much faster, thereby shifting left the overall verification project cycle.

	Default simulation UVM error count per regression	Simulation using AI/ML UVM error count per regression
Regression 1	67	84
Regression 2	64	86
Regression 3	70	79
Regression 4	72	111
TOTAL 4 regressions	**TOTAL=273**	**TOTAL=360**

Inconsistent constraints: **0**
Scoreboard mismatch errors: **0**

Inconsistent constraints: **89**
Scoreboard mismatch errors: **7***

Figure 1: Rare testbench issues exposed using AI/ML on a stable testbench

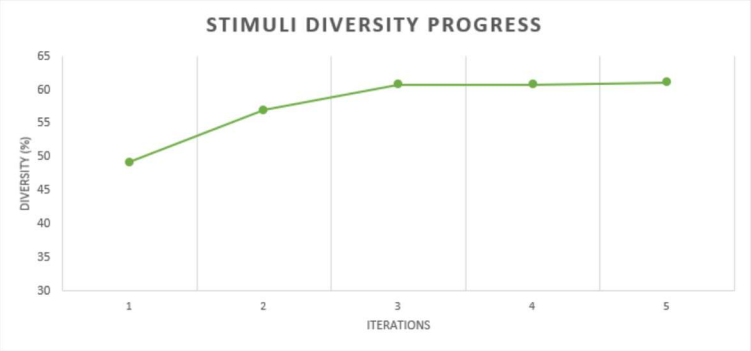


Figure 2: Stimuli Diversity Progress

In this paper, we propose a methodology for adopting AI/ML techniques (such as ICO) early in the project cycle. In Figure 2, we illustrate the progress of diversity (a proxy for stimuli quality) while running back-to-back regressions. The diversity in values generated is computed as a percentage of Shannon entropy [4] achieved over the maximum possible diversity. In practice, achievable diversity can be limited by constraint tightness.

The first step is to determine the eligibility of a given testbench suitable for AI/ML application. This is done using a prognosis report that indicates whether the testbench has sufficient freedom available to diversify and bias the random variables in the testbench using AI/ML. If the constraints are more directed in nature, then they can be considered tighter and have less freedom to bias as opposed to constraints that are less directed. The constraints with more freedom (i.e., capturing larger stimuli space) enable AI/ML to improve the diversification of the stimuli space. Once the testbench is determined suitable, we proceed with using the AI/ML application on the entire regression. This methodology proved to be extremely helpful in uncovering testbench latent issues, hard-to-hit scenarios, and exposing bugs early on.

This AI/ML technology provides a qualitative assessment of testbench randomness and the complexity of constraints. This assessment gives us a base on which we can proceed to enable the AI/ML application for that testbench. In Figure 3, we present the variable randomness and value randomness as percentages and constraint density (i.e., the average number of constraints per variable) for the *Cache testbench*. Similarly, we present data for the *Render Block testbench* in Figure 4. Higher values show which testbenches are suitable for AI/ML applications as described in the next section.

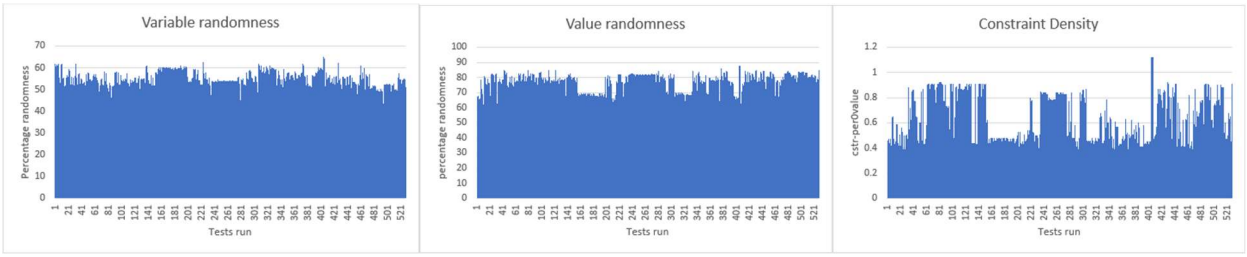


Figure 3: *Cache* testbench randomness & complexity (a) Variable Randomness (b) Value Randomness (c) Constraint Density

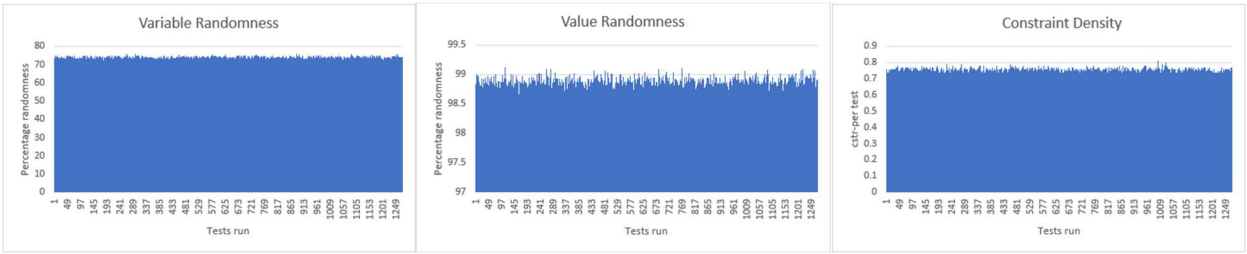


Figure 4: *Render Block* testbench randomness & complexity (a) Variable Randomness (b) Value Randomness (c) Constraint Density

III. RESULTS

Overview

We will first present an overview of the results observed for 3 projects: A, B, and C, comparing the use of AI/ML techniques in each project versus without using AI/ML. We will then discuss each project in more detail later.

For project A, we applied AI/ML solutions in the late stage of the project when the technology was made available to us. Surprisingly, even for the late stage of the project in which the testbench and DUT are considered stable, we observed 30% more bugs in both testbench and DUT (Figure 1). Moreover, we witnessed a 40% reduction in the number of regressions needed to achieve the targeted coverage. Encouraged by the results, we decided to adopt AI/ML solution early in project B.

We intercepted project B's execution with AI/ML solution and immediately started witnessing a positive impact. As shown in Figure 5, we were able to uncover most of the bugs in the first 4 weeks as compared to without adopting the proposed methodology Note, in the X-axis, we show the bugs exposed per week relative to the starting week of the two approaches respectively. We explained our experimental setup in more detail later.

For project C, as shown in Figure 6, we compared the functional coverage rate with and without the proposed methodology on the same block in a parallel setup With improved diversified stimuli, we were able to achieve faster coverage in fewer iterations. This helped us to write fewer directed tests, thereby saving 2 weeks of manual effort. It also helped us to optimize the regression size, enabling a 15% reduction per block of grid resource usage.

We observed that AI/ML guided us to shift-left functional verification by several weeks consistently across projects.

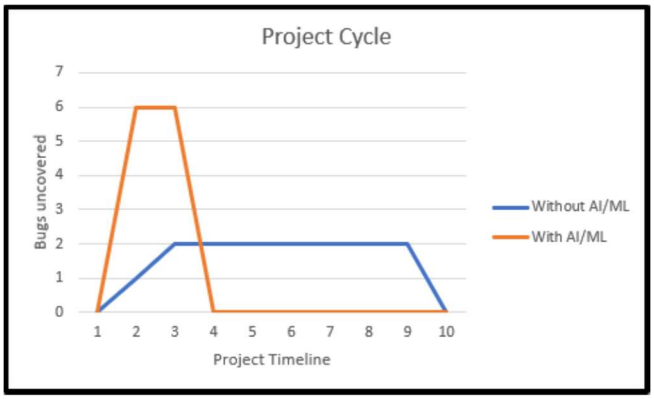


Figure 5: Shift-left project timeline with early use of AI/ML

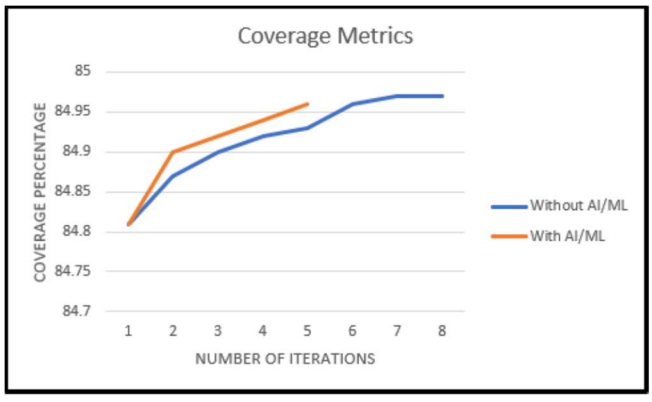


Figure 6: Accelerating Coverage rate with early use of AI/ML

Project A – L2 Cache Block details

We provide an overview of the Cache block inside a graphics IP, as shown in Figure 7. The Cache test bench is implemented using constraint randomization, where the entire function and checkers are modeled using a UVM-based methodology. The Cache is a high-performance memory with high efficiency and low latency. The Cache design also handles the entire GPU Pipeline architecture synchronization functionality. A high-level test bench architecture diagram is shown below.

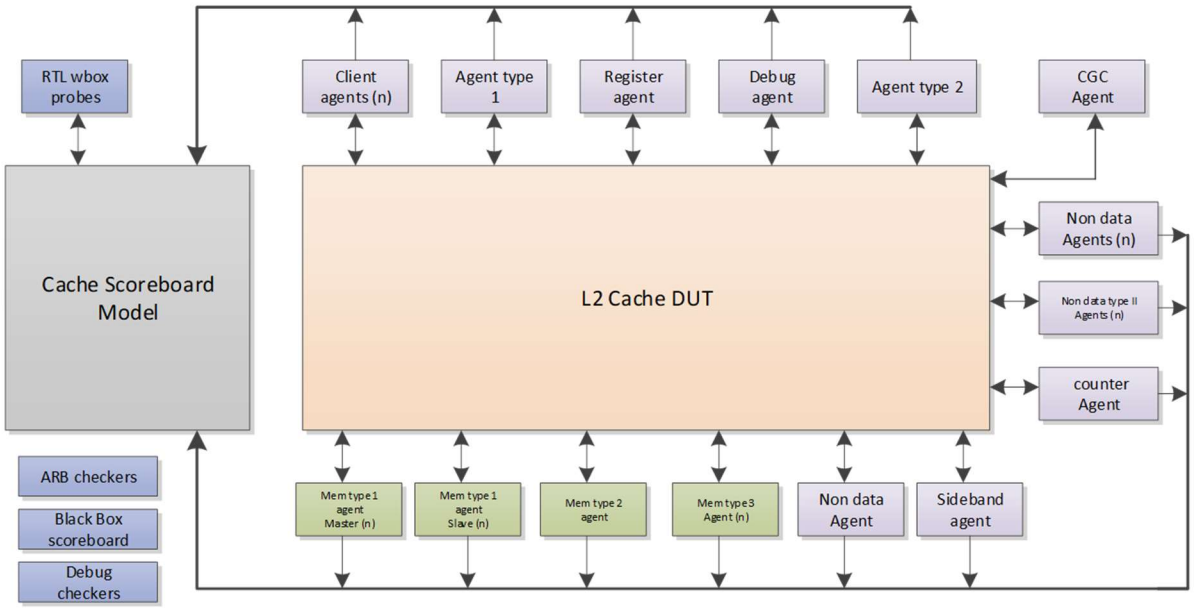


Figure 7: L2 Cache Test Bench Environment

When we started the evaluation of applying AI/ML, the Cache feature set and design development was completed and the test bench environment was considered mature and stable. The pass rate for the test bench was above 98%. Furthermore, the code and functional coverage was around 95%. A total of 10 regressions with and without the AI/ML solution were tried on the Cache block. We found significant improvement in verification results using AI/ML solutions as summarized below:

- Exposed scenarios that are not captured as part of functional coverage (Figure 1). Some examples are:
    - *Testbench failure #1: Constraint inconsistency failures*
    - *Testbench failure #2: Design SVA failures due to issues in the testbench UVM driver*
    - *Testbench failure #3: Scoreboard issues related to the checker*
    - *RTL failure: Design deadlock scenarios*
- Full insight into the input stimulus distribution such as,
    - *Under-constraint vs. Over-constraint scenarios*
- Faster Coverage Closure
    - *Achieved 95% functional coverage in 6 regressions vs. 10 regressions. Savings of 4 regressions and IT resources*.

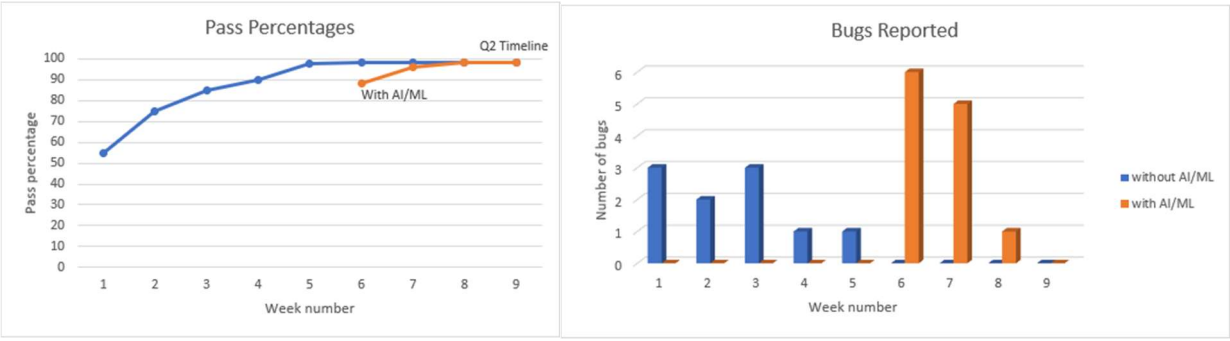
Project B – Modified L2 Cache details

In project B, we provide our observations using the AI/ML solution on a new Cache design. This was a time-critical project with a shorter runtime compared to other projects. When we ran random regressions daily without AI/ML technology enabled, we would see 2 to 3 design bugs a week. At 4 to 5 weeks, we observed the feature bring-up and scoreboard model were becoming stable, and the random regression pass rates were nearly 98%. Encouraged by the positive impact of the AI/ML solution in project A, we decided to intercept this project at week 6 to enable AI/ML solution to reconfirm if it can provide additional value. We were truly impressed with the results of shift-left in bug discovery in this time-critical project, as shown in (Table 1, Figure 8). We summarize our observations as follows:

- Early bug discovery (in both testbench and DUT) and quickly hitting rare corner bugs (Table 1, Figure 8)
- Shift-left in functional coverage by 30% when coverage is greater than 90%. Like project A, we achieved the same functional coverage in 6-7 iterations with the AI/ML solution when compared to 10 iterations without the AI/ML solution.
- A 10-15% savings of grid or compute resources (Figure 9) (explained later)

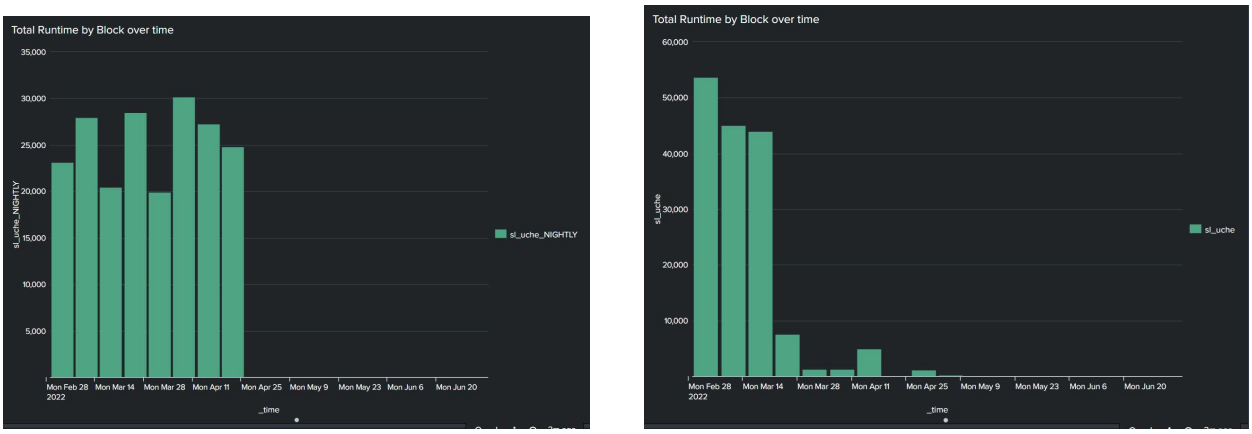
Table 1: Bug rate using with and without AI/ML solution

Week	Pass %	AI/ML Enabled?	# Testbench bugs	# RTL bugs
Week 1	55%	no	13	3
Week 2	75%	no	10	2
Week 3	85%	no	11	3
Week 4	90%	no	11	1
Week 5	98%	no	7	1
Week 6	88%	Yes	18	6
Week 7	96%	Yes	7	5
Week 8	98.50%	Yes	5	1
Week 9	98.50%	Yes	5	0



(a) (b)

Figure 8: (a) Pass percentages (b) Bug rate with and without using AI/ML solution



(a) default (w/o AI/ML) (b) with AI/ML

Figure 9: Total grid runtime per block over week. (a) Default 198K time units vs. (b) AI/ML 162K time units for the same coverage

Ultimately, we expect ROI on adopting this new technology to help us reduce verification costs and improve productivity. To measure this quantitively, we compared the grid compute usage with and without using AI/ML for a block as illustrated in Figure 9. The total runtime of the simulation jobs per block was compared without (default) and with AI/ML, to achieve the same coverage goal as one of the cost metrics to highlight the magnitude of the savings achieved. Figure 9(a) shows the default requirement of 198K time units (area of the bar chart) versus the AI/ML solution requiring a total of 162K time units, a reduction of nearly 18%. Note that the default regression was conducted by automated scripts, while the AI/ML regressions were conducted by the custom ad-hoc scripts executed by multiple users. This explains the non-uniform usage in AI/ML regressions.

Project C – Render Block details

In project C, we present the impact of the AI/ML solution on the Render Block where we experimented with and without the AI/ML solution in a parallel setup on the same Block version. This block provides blending, color, and depth calculations in the graphics pipeline architecture. The number of color formats, depth formats, and title packing combinations with different register programming combinations make this design block very complex and very hard for the design verification team to cover all the scenarios. This block has approximately 350K coverage bins corresponding to various scenarios corresponding to different color formats, depth formats, and tile packing. Our constraints random test benches are designed appropriately to cover all the combinations, but that would need several months of back-to-back regressions to achieve acceptable code and functional coverage. As we get close to the final project milestone to meet the coverage closure target, we needed to put in a lot of manual effort, such as tweaking constraints, to create certain directed test cases or to chain the delay profiles on each interface. All these efforts take a week or more to cover the remaining 200 to 500 bins, based on how good input stimuli are generated.

Table 2: Code and Functional Coverage with and without AI/ML solution on Render Block

Render Block Coverage	Metric	Default	AI/ML Enabled	Improvement with AI/ML
Code	Line	88064	89231	1167
	FSM	855	885	30
	Condition	356836	359019	2183
	Branch	66888	69931	3043
Functional	Cover Points	1757	1804	47
	Coverpoint bins	39628	39969	341
	Cross Coverage bins	305445	306859	1414
	Total Bins	347984	349661	1677

After enabling AI/ML technology in random regressions for this block, we witnessed dramatic improvement or shift-left in functional coverage closure by 1.5 weeks. We observed that most of the complex bins which needed tweaking of constraints or delay profile automatically are now getting hit quickly. Consequently, we now need to spend very minimal effort (maybe for less than 100 bins) on directed /constraint tweaking or delay profile tweaking. Table 2 shows the coverage metrics with and without using the AI/ML solution. Figure 6 shows the coverage acceleration with and without the AI/ML solution.

IV. CONCLUSION

In this paper, we focused on using an AI/ML solution for improving stimuli quality early in our project, thereby achieving faster testbench stabilization by discovering hard-to-hit testbench bugs due to under- and over-constraints and fixing them early on. Faster testbench stabilization early on has multiple benefits, such as

- finding early design bugs which are less costly to fix,
- reducing the manual effort required in writing directed test cases,
- reducing the regression size,
- removing redundant tests,
- finding potentially omitted scenarios that may lead to bug escapes or coverage holes,
- make the testbench conducive to faster functional coverage closure.

We presented the verification impact of using the AI/ML solution on three projects. Then we showed how we can achieve improved productivity while reducing verification costs by adopting such technology early in our project cycle. This productivity and savings were achieved without putting any extra manual effort to change or rewrite any part of our functional coverage models.

REFERENCES

[1] "Intelligently Optimizing Constrained Random", https://semiwiki.com/artificial-intelligence/314806-intelligently-optimizing-constrained-random/

[2] “Intelligent Coverage Optimization: Verification Closure In Hyperdrive", https://semiengineering.com/intelligent-coverage-optimization-verification-closure-in-hyperdrive

[3] “Accelerate Coverage Closure Using AI/ML-based VCS ICO Technology", https://www.synopsys.com/verification/resources/webinars/vcs-ico-pt2.html

[4] C. F. Shannon and W. Weaver: Mathematical Theory of Communication. University of Illinois Press, Urbana (1949).