AI - accelerating coverage closure using intelligent stimulus generation

Jainender Kumar (Samsung-India), Ronak Bhatt (Samsung-India), Garima Srivastava (Samsung-India), Ashutosh Sinha (Cadence-India), Prashant Teotia (Cadence-India)

***Abstract*- As the number of Intellectual Property (IP) cores in modern System-on-Chip (SoC) designs continues to grow, coupled with the increasing trend of multi-chip designs, the complexity of the bus matrix has significantly evolved. Interconnects are implemented across various layers to meet timing requirements and support many initiators and target ports with mixed protocols. Consequently, functional coverage has become a critical aspect of SoC verification, particularly for ensuring comprehensive validation of interconnects and their data paths. However, achieving functional coverage for such complex SoCs, which often involve an enormous number of bins—sometimes nearing a million—presents several challenges. For instance, an SoC with 150 initiators and 500 targets must cover the cross-product of parameters like initiator, target, size, length, burst, and others as applicable. This complexity can result in more than half a million bins, complicating their management, analysis, and reporting, necessitating substantial computational resources and advanced tools.**

To achieve full coverage for these bins in a complex SoC, thousands of iterations and extensive manual efforts are required. Moreover, after a certain point, when coverage saturation is reached, randomized input stimulus fails to provide significant incremental coverage in successive iterations. This occurs because the current randomization engine repeatedly hits the same bins, leading to diminishing returns in coverage progress. Consequently, this prolongs verification closure timelines and can adversely impact time-to-market. To address this issue, adopting artificial intelligence (AI) and machine learning (ML) technologies in the design verification (DV) environment is crucial. This paper proposes leveraging AI/ML-based technology using "input-bias" feature of Cadence Xcelium SimAI tool to accelerate functional coverage convergence while preserving the randomness of the stimulus. In the proposed approach, an input bias file is derived based on each initiator constraints and its corresponding accessible targets. This file is provided as input to the AI tool, which utilizes it to monitor coverage. Based on feedback, the AI tool determines which input constraints should be given higher weightage, ensuring that new cover groups and bins are targeted in subsequent iterations instead of repeatedly hitting the same bins. This approach not only prevents coverage saturation, but also enables faster coverage achievement in fewer iterations. Additionally, it facilitates the coverage of cross bins and the identification of corner-case scenarios at an early stage, significantly enhancing the efficiency and effectiveness of SoC verification.

I. INTRODUCTION

In modern SoC design verification, achieving comprehensive functional coverage efficiently is crucial to gaining confidence in overall SoC verification. Functional coverage, including cover groups and cover bins, plays a critical role in ensuring that various routes and scenarios within the interconnect are thoroughly verified. To handle today's complex SoCs, more sophisticated and structured interconnect designs are employed to ensure proper timing closure and functional correctness.

Specifically, in data route verification, it is essential to verify that the communication between different IP blocks and subsystems function as expected under all conditions. The sheer number of corner bins and coverage groups is growing exponentially, leading to significant challenges in achieving comprehensive verification. Furthermore, with coherent interconnects, it is necessary that initiators (coherent or non-coherent) and targets (DRAM, memory, SFR etc.) support various protocol configurations. In such scenarios, cover points that involve crosses of parameters such as source, destination, transaction type, data size, request type, and length result in a massive number of cover bins that must be covered within a limited timeframe.

Covering such an extensive number of coverage groups becomes a bottleneck, making it impractical to cover all corner cases due to constraints on time and resources. Additionally, achieving coverage closure requires running many test cases With an increasing number of cover bins to hit, simulation times become prohibitively long, especially for large and complex SoCs. Moreover, large SoC designs demand significant memory and computational power to store and track coverage information across all cover groups and bins, leading to resource constraints in simulation farms operating under constrained environments. Limited resource availability makes exhaustively simulations for large designs infeasible, resulting in performance issues and extended verification timelines. Some corner cases may remain unreached during regular simulation runs, either due to the rarity of the conditions required to trigger them or because redundant coverage bins consume valuable simulation time.

To address these challenges, an approach is required to achieve faster coverage closure within the verification timelines. The proposed method accelerates functional coverage by leveraging advanced technologies and

methodologies, such as artificial intelligence (AI) and machine learning (ML). The paper specifically utilizes the **"input-bias"** feature of the Cadence Xcelium SimAI tool. This method effectively analyzes a large set of coverage bins (over 600K) and biases input constraints based on feedback, thereby improving both speed and accuracy.

The proposed solution has been applied to real designs, resulting in a 30% reduction in overall bus verification closure time. This method uses iterative evaluation to maximize the coverage and achieve faster closure with fewer regression runs compared to the traditional approach.

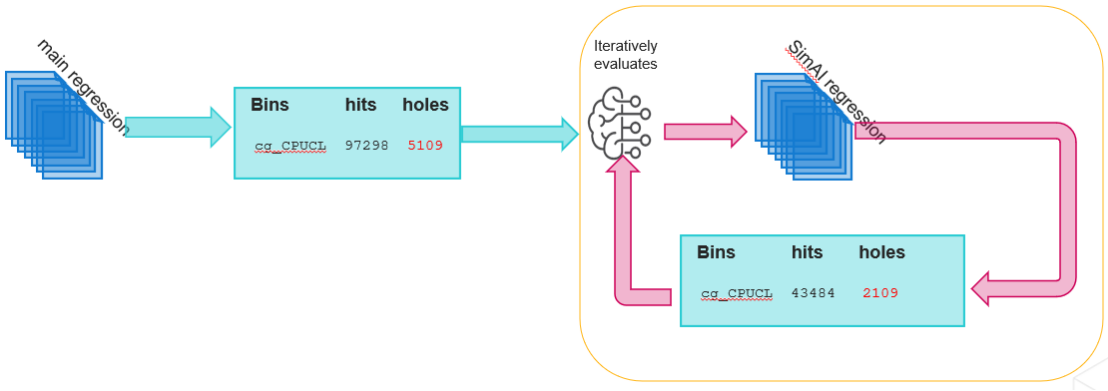

Figure 1 Multi-chip connected via UCIe in a typical

On each iteration, the AI engine evaluates and identifies coverage holes. The machine learning model is then updated with information about its performance. The basic flow is to run random regression with coverage and randomization data. When coverage becomes largely saturated and is growing slowly, then use the AI tool to target coverage holes more efficiently. The AI tool can be trained at any phase of regression, using the previous run as the base regression session. After training, the toll iteratively repeats simulation runs to target coverage holes until coverage is fully closed. The key goal of the AI tool is not just to regain coverage, but to increase the probability of hitting coverage holes. During the coverage closure phase, the AI tool performs full random runs to stress the design in various ways while targeting coverage holes.

II. SoC COVER GROUP STRUCTURE

For complex SoCs, it is crucial to ensure that combinations of source IP, destination IP, transaction types (read/write), data sizes, lengths, etc., are fully exercised. In such cases, cross-coverage tracks whether all combinations of these variables are covered. This is vital for testing SoCs with various subsystems connected through multiple levels of interconnect, supporting different protocols.

However, as the number of cross-coverage bins increases, the combinations grow exponentially, leading to challenges such as longer simulation times and potential coverage gaps. Despite these challenges, cross-coverage is essential for the comprehensive verification of complex SoC designs.

Figure 2 shows some of the possible combinations of cover groups for an AXI initiator accessing multiple targets, with constraints defined for each target.

To create such extensive coverage groups for a complete SoC, a Perl script is used. Based on input files, containing initiator constraints, target address space, and initiator-target reachability, the script generates cover groups for each initiator targeting all the accessible targets. Additionally, the script generates monitor connection files, simplifying integration into any UVM-based framework environment. The script is also capable of ignoring illegal bins or invalid bins during the process. Figure 3 illustrates this workflow.

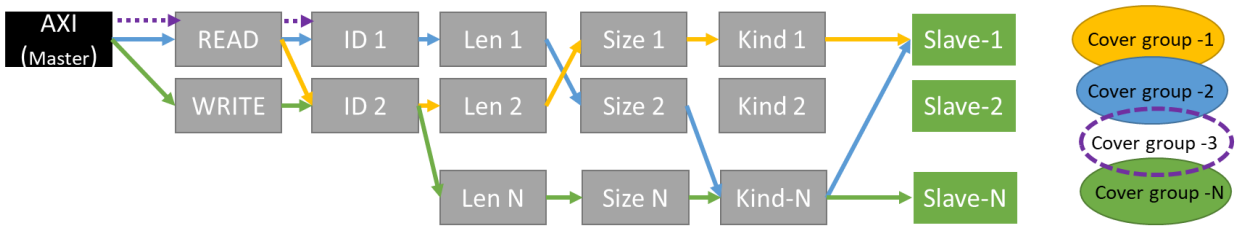

Figure 2 Cover group formation for AXI initiator in an interconnect and corresponding similar cover group definition

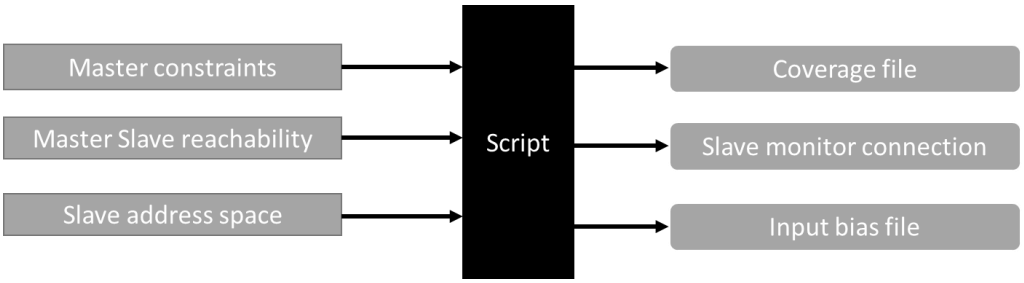

Figure 3 Script utilizing input files to generate plug n play coverage files.

TABLE I
COVERAGE BINS ANALYSIS FOR VARIOUS SOCS

Parameter	SoC 1	SoC 2	SoC 3	SoC 4
No. of Initiators	134	149	143	202
No. of targets	400	383	407	598
No. of cover bins	1033K	604K	594K	1659K

The script maintains a initiator database and a target database to simplify the generation of cover groups for each initiator, as shown in Figure 4 and Figure 5, respectively. These databases contain various fields that are updated using the information provided in the input files, such as initiator constraints, reachability information, and target address range. For a particular initiator database, the fields can be added or modified as required.

For example, in the case of the APB protocol, the target list is divided into active APB target (Slave_List_APB_A) and passive APB target (Slave_List_APB_P). This distinction is helpful while creating a cover group. For active targets, an initiator can have the possible direction of **READ** and **WRITE** whereas for passive targets, the possible direction is limited to READ. This ensures that unnecessary WRITE operations to the passive targets, which can potentially alter the behaviour of the RTL, are avoided. Maintaining such a list also helps to prevent the generation of illegal crosses.

```
$info{$cell[$info{'Master_idx_info'}{'Interface Name'}]} = {
          'Port'   => 'Master',
          'Direction'  => $cell[$info{'Master_idx_info'}{'Direction'}],
          'D_Width'  => $cell[$info{'Master_idx_info'}{'Data Width'}]/8,
          'Protocol'  => substr($cell[$info{'Master_idx_info'}{'Protocol'}],0,3),
          'RID'   => $cell[$info{'Master_idx_info'}{'Read ID Width'}],
          'WID'   => $cell[$info{'Master_idx_info'}{'Write ID Width'}],
          'Slave_List' => '',
          'Slave_List_AXI' => '',
          'Slave_List_AXI_64b' => '',
          'Slave_List_AHB' => '',
          'Slave_List_APB_A' => '',
          'Slave_List_APB_P' => '',

        },
```

Figure 4 Master database format utilized by the Script

```
$info{$cell[$info{'Slave_idx_info'}{'Interface Name'}]} = {
          'Port'   => 'Slave',
          'Direction' => $cell[$info{'Slave_idx_info'}{'Direction'}],
          'Protocol' => substr($cell[$info{'Slave_idx_info'}{'Protocol'}],0,3),
          'A_Agent' => ($cell[$info{'Slave_idx_info'}{'Attach Active Agent'}] eq 'Y')?1:0,
          'Within_64b' => 1,
        };
```

Figure 5 Slave database format utilized by the Script

The coverage file output from the script will have a coverage group definition for each of the initiators as given below.

```
covergroup cov_INITIATOR_M0;
        option.per_instance = 1;
        Read_IdTag : coverpoint INITIATOR_M0_item.IdTag
        {
                bins rid_value = {[0:15]};
        }
        BurstKind : coverpoint INITIATOR_M0_item.Kind
        {
                bins kind[] = {INCR,WRAP,FIXED};
        }
        BurstSize : coverpoint INITIATOR_M0_item.Size
        {
                bins size[] = {DENALI_CDN_AXI_TRANSFERSIZE_FOUR_WORDS};
        }
        ...
        cp_INITIATOR_M0_direction_X_read_IdTag: cross Read_IdTag, Read_Direction ;
        cp_INITIATOR_M0_direction_X_write_IdTag: cross Write_IdTag, Write_Direction ;
        cp_INITIATOR_M0_X_TARGET_1: cross BurstLength, BurstKind, BurstSize, Direction iff (target_target == TARGET_1);
        cp_INITIATOR_M0_X_TARGET_2: cross BurstLength, BurstKind, BurstSize, Direction iff (target_target == TARGET_2);
        cp_INITIATOR_M0_X_TARGET_3: cross BurstLength, BurstKind, BurstSize, Direction iff (target_target == TARGET_3);
        cp_INITIATOR_M0_X_TARGET_4: cross BurstLength, BurstKind, BurstSize, Direction iff (target_target == TARGET_4);
        cp_INITIATOR_M0_X_TARGET_5: cross BurstLength, BurstKind, BurstSize, Direction iff (target_target == TARGET_5);
                ...
```

Depending on the project, valid cover groups are identified and listed in Figure 2. It is important to note that the number of cover bins increases 3-4x when the same die is implemented in a multi-die setup further complicating coverage closure.

III. AI SETUP IN SIMULATION

The AI framework uses randomization and coverage data from the design and testbench to create a machine-learning model that accelerates regression performance. This model learns from iterative regression sessions and generates optimized regressions based on predefined criteria, such as stressing specific design instances, minimizing directed tests, or maximizing regression throughput using random tests without committing seeds. The complete flow involves a series of steps that uses an existing regression (or simulation runs) as input, generate learning models, and synthesize an optimized version. Using this simplified flow, you can:

- Apply machine learning
- Synthesize a regression
- Run the synthesized regression

A. Machine Learning

To implement a regression for machine learning, the simulator uses appropriate switches to analyse all the random variables during regression runs. This step creates a base regression run that serves as input for the AI tool to analyse and build a learning model. Any regression can be used as the base regression run, regardless of coverage levels.

At the end of each base regression run, the AI tool generates several files containing configuration and path information, coverage bit-vectors, and randomization data necessary for regression synthesis.

B. Synthesizing a regression

During the synthesis phase, the AI tool enables an iterative learning flow framework, allowing it to repeatedly learn from the original input regression and other saved data. Within this framework, users can explicitly define use cases and convergence policies. For example, users can enable coverage maximization, directing the AI tool to synthesize a regression to target uncovered bins from the original regression.

Running a synthesized regression helps generate scenarios targeting corner cases or other critical areas, improving coverage of error conditions. It can also be used for bug-hunting runs, filling a CPU budget, or targeting challenging bins, including cross-coverage bins requiring multiple runs.

Finally, the AI tool generates a synthesized regression based on the base regression, which may include small, optimal, or exploratory regression runs. It saves datasets, learning models, synthesized regressions, and reports in a directory, which are used in subsequent iterations of learning.

C. Run the synthesized regression

The synthesized run from the earlier setup is then utilized with a default learning database to cover random variables across all possible cover group information. To refine convergence, a one-time input bias file is provided, containing cross-coverage definitions and user-specified random variables. This file enables the AI tool to analyse regression outcomes in line with the input bias structure. With each new regression run, the learning matrix is updated, allowing the AI tool to evaluate the impact of control knobs.

The input bias file specifies valid bins for each initiator and accessible target, avoiding invalid bins that could degrade performance by wasting time attempting to close them. The input bias file is also generated using the same Perl script as discussed above. Below is an example of file content that is coverage-aware.

```
[
{  "axiTransactionUsr::initiator_port_name": INITIATOR_M0  },
{  "target_id_class::target_initiator_idx":  [83]  },
{  "axiTransactionUsr::target_name":  TARGET_1  },
{  "target_id_class::target_target_idx":  [210 ]  },
{  "denaliCdn_axiTransaction::Direction": [1,2]  },
{  "denaliCdn_axiTransaction::Length": [1]  },
{  "denaliCdn_axiTransaction::Kind": [2]  },
{  "denaliCdn_axiTransaction::Size": [1,2,4,8]  }
]
```

IV. FEATURES

The use of AI/ML tools for functional coverage offers the following features:

A. Automatic constraints learning

The AI tool generates a learning model by analyzing all random variables in each base/main regression set. This model includes constraint variable information for all possible inputs stimulating the initiator in the design. The AI tool updates the learning model with each regression run, ensuring it adapts to the latest constraints scenarios.

B. Intelligent feedback mechanism and coverage gap detection

The AI tool automatically analysis synthesized regression runs and compares them with the learning model from the previous runs to identify cover group patterns with a zero-hit ratio. Using coverage gap information, the AI tool recognizes uncovered cover groups and determines the convergence strategy for subsequent regression runs. Automatic coverage gap detection is especially critical for managing design with hundreds of thousands of bins.

C. Optimized Stimulus generation:

The input bias constraint file, generated from the cover group model, provides the AI/ML tool with visibility into the relationships between constraints and associated cover groups. The AI tool analyzes this information and biases the input stimulus to utilize specific random values effectively.

D. Intelligent test stimulus

Based on the input basis file, the AI tool eliminates redundant tests that do not contribute to additional coverage. Subsequent runs are more targeted at uncovered bins while maintaining randomization, ensuring efficient resource utilization.

V. ADVANTAGES

The implementation of AI/ML-based coverage closure streamlines functional coverage achievement and offers the following benefits:

A. Dynamic Stimulus adjustment

The proposed approach allows input bias stimulus to be dynamically adjusted at any stage of the coverage activity based on the base regression set. This feature enables seamless utilization of both generic test vectors and AI-synthesized test vectors, making the approach robust and adaptable.

B. Easy implementation.

Integrating AI/ML into the regression setup is a one-time effort performed at the start. The setup is easily portable to other projects with minimal changes. Furthermore, the automatic generation of input bias file makes it independent of the number of bins or the scale of the design, allowing effortless implementation in any SoC.

C. Scalable approach

The method is highly scalable and independent of design architecture. It is future-ready and can be implemented in multi-die architecture with minimal additional effort. The AI tool demonstrates superior performance for larger designs, offering high coverage efficiency.

D. Efficient resource utilization

By optimizing test vectors and synthesizing regressions, the AI tool significantly reduces the number of iterations required to achieve coverage closure. This compressed regression approach maximizes overage while minimizing simulation time, computer resources, and memory requirements.

E. Reduced Time-to-Market

AI/ML-based coverage-driven verification enables early analysis of corner-case cover groups and early detection of design bugs, significantly accelerating the time-to-market for products.

VI. RESULTS

When applied to a real design, the following results were achieved.

A typical mobile platform fabric with approximately 100 initiators and 380 targets, targeting ~600 K bins, achieved saturation at 98%. After saturation, ~8.4K simulation hours of regression runs were required to achieve 100% coverage. However, using the proposed methodology, the time to achieve 100% coverage was reduced to ~3.8K simulation hours.

It was observed that the AI tool delivers better results when trained early in the regression phase. For a typical hard-to-hit cover group, achieving the AI tool with the input bias solution showed significant improvement.

Initiator: ~100 Targets: ~380
Target coverage bins: > 300,000
Original Regression > 8400Hrs (~98% coverage)
Input Bias: ~3800 Hrs (100% coverage)

TABLE II
IMPROVEMENT FOR COVERAGE TO 100% USING AI TOOL

Original		**AI synthesized run**		**Sim time reduction**
Coverage	**Original run**	**Coverage**	**Original run**	
89% to 98%	~8.4K	89% to 100%	~3.8K	> 54%
46% to 88%	~8.7K	46% to 100%	~3.8K	> 56%

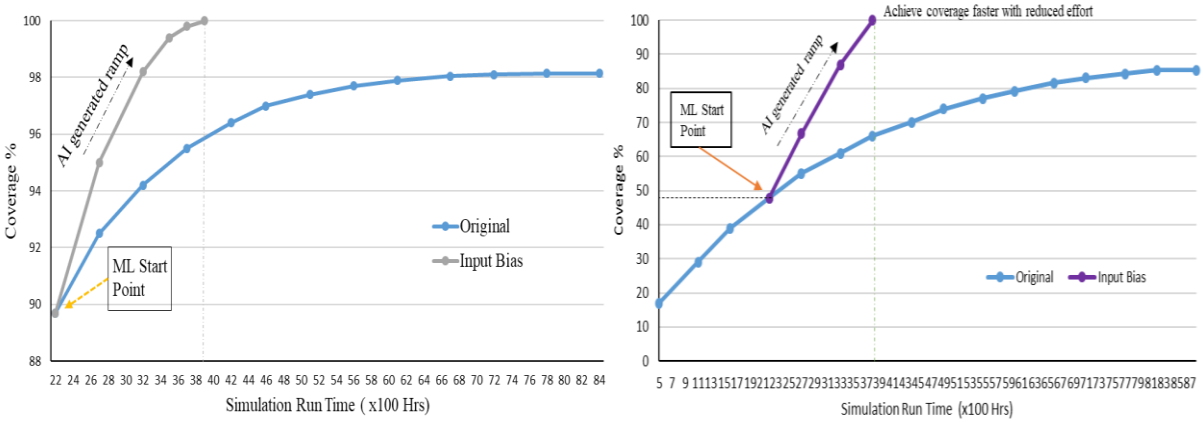

Figure 6. AI generated coverage vs normal coverage after applying ML at 88% and 46% respectively.

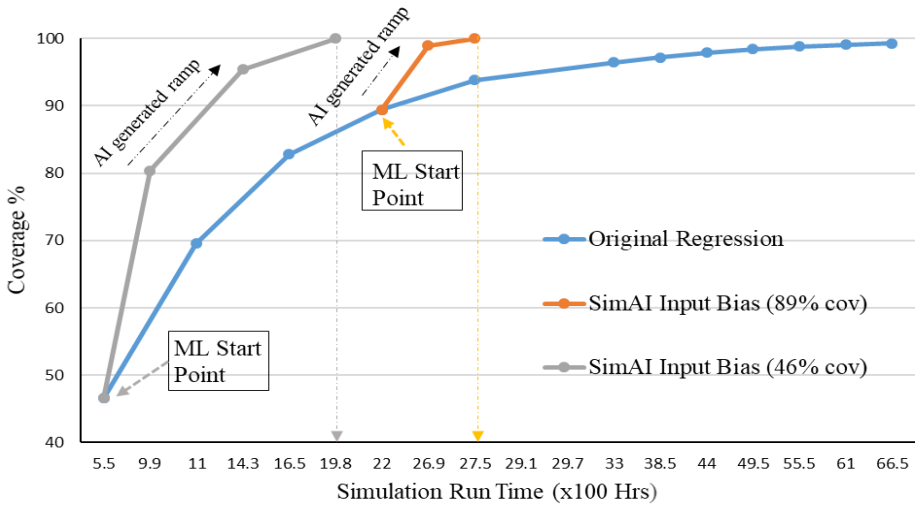

Figure 7. AI-generated coverage vs normal coverage applied on the same regression with different base regression

Figure 7 shows a graph when applying AI to regression at an early stage of regressions. It is evident that higher gains are observed when AI is employed earlier.

Covergroup: cov_wlbt_core_x_active_targets
Initiator: wlbt_core Targets: ~380
Target coverage bins: 9250
Original Regression > 8700Hrs (~85% coverage)
Input Bias: ~3800 Hrs (100% coverage)

VII. CONCLUSION

The AI/ML approach has been implemented in various complex SoCs with coverage bins ranging from 0.7 million to 1 million. Under normal randomization stimulus, coverage typically saturates in the final stages, requiring more iterations to complete. In contrast, the input bias approach dynamically adjusts stimuli based on non-hit bins after each iteration, ensuring a constant increase in coverage. This dynamic adjustment prevents saturation and achieves 100% coverage closure in at least 30% less time. By leveraging AI/ML methodologies, this approach maximizes coverage with a reduced regression set, irrespective of the number of bins in a design.

REFERENCES

[1] Mammo, B. Reining in the Functional Verification of Complex Processor Designs with Automation, Prioritization, and Approximation. 2017. Available online: https://deepblue.lib.umich.edu/bitstream/handle/2027.42/137057/birukw_1.pdf (accessed on 15 May 2024).
[2] Siemens. Available online: https://blogs.sw.siemens.com/verificationhorizons/2021/01/06/part-8-the-2020-wilson-research-group-functional-verification-study/ (accessed on 15 May 2024).
[3] Truong, A.; Hellström, D.; Duque, H.; Viklund, L. Clustering and Classification of UVM Test Failures Using Machine Learning Techniques. In Proceedings of the Design and Verification Conference (DVCON), San Jose, CA, USA, 26 February–1 March 2018.
[4] El Mandouh, E.; Maher, L.; Ahmed, M.; ElSharnoby, Y.; Wassal, A.G. Guiding Functional Verification Regression Analysis Using Machine Learning and Big Data Methods. In Proceedings of the Design and Verification Conference and Exhibition Europe (DVCon), Munchen, Germany, 25 September 2018.
[5] Ismail, K.A.; Ghany, M.A.A.E. Survey on Machine Learning Algorithms Enhancing the Functional Verification Process. Electronics 2021, 10, 2688.
[6] Varambally, B.S.; Sehgal, N. Optimising Design Verification Using Machine Learning. An Open-Source Solution. ArXiv 2020, arXiv:2012.02453.
[7] Menzies, T.; Pecheur, C. Verification and Validation and Artificial Intelligence. Adv. Comput. 2005, 65, 153–201.