Accelerating Coverage Directed Test Generation for Functional Verification: A Neural Network-based Framework

Fanchao Wang, Hanbin Zhu, Pranjay Popli, Yao Xiao, Paul Bodgan and Shahin Nazarian
University of Southern California
Los Angeles, CA, USA
{fanchaow,hanbinzh,ppopli,xiaoyao,pbogdan,shahin}@usc.edu

ABSTRACT

With increasing design complexity, the correlation between test transactions and functional properties becomes non-intuitive, hence impacting the reliability of test generation. This paper presents a modified coverage directed test generation based on an Artificial Neural Network (ANN). The ANN extracts features of test transactions and only those which are learned to be critical, will be sent to the design under verification. Furthermore, the priority of coverage groups is dynamically learned based on the previous test iterations. With ANN-based screening, low-coverage or redundant assertions will be filtered out, which helps accelerate the verification process. This allows our framework to learn from the results of the previous vectors and use that knowledge to select the following test vectors. Our experimental results confirm that our learning-based framework can improve the speed of existing function verification techniques by 24.5x and also also deliver assertion coverage improvement, ranging from 4.3x to 28.9x, compared to traditional coverage directed test generation, implemented in UVM.

1 INTRODUCTION

Gordon Moore's [12] observation that the number of transistors in modern chips doubles approximately every 2 years, has also resulted in exponential escalation of the complexity of modern technologies and systems. Furthermore, with design specifications becoming elusive, which leads to disagreement among designers on implementation of the functional models in HDL. Upon consensus of a system design, RTL designers translate specifications into millions of lines in RTL code, which makes it difficult to verify all possible cases a user might encounter. In addition to misinterpretation of spec, the design flow, namely synthesis and physical design and optimization steps may introduce more functional bugs.

As the complexity of designs grows, systematic functional verification plays a more significant role in making chips functionally reliable for users. It is not feasible to exhaustively simulate every possible case and state of the system. For example, a typical 64-bit adder calls for 2^{129} possible test vectors to cover all corner cases. Assuming a 2 GHz processor is used to run it and one stimulus per 16 cycles since the depth of modern pipeline is 15-20, roughly 2^{76} years are needed to verify a simple adder. On the other hand, formal verification techniques, necessitate a detailed understanding of the design well ahead of time and also may suffer from state explosion. Therefore, compared to the design of an actual chip, verification adds more energy, cost, complexity, and especially time [6]. Traditionally, a combination of formal verification, and constrained-random simulation-based has been used in literature and industry to mitigate the cost of verification and achieve high coverage. However the increasing complexity of systems with larger number of inputs, states, interfaces and protocols, as well as tighter time-to-market enforced by market, has led the designers to seek beyond traditional verification algorithms towards machine learning algorithms to mitigate the verification challenges.

In the simulation-based verification, test stimuli are randomly generated under spec constraints and applied to DUV (Design Under Verification). CDG (Coverage-directed Test Generation) can be incorporated to increase the coverage rate within the limited time. However the traditional CDG can underperform in progressively complex environments due to manual bias. For example, in state-of-the-art micro-architectures, features such as dynamic scheduling with out-of-order executions and dynamic branch prediction in multi-core designs, would make it difficult to predict what constraints would produce tests with high coverage [8][9].

In this work, we aim to revisit the problem of verification of ASIC blocks that are typically complex RTL blocks with high chance of error bugs, due to complex functional specifications, as well as large number of transistors and wires, and parameters associated with them, that may impact the functionality. We understand many of those parameters may turn out to be irrelevant, therefore we aim to design a verification platform that automatically learns the important dependencies and utilizes the knowledge it gains from previous test failures in order to increase the coverage and also the speed of verification. To guide test generation toward high coverage rate, we introduce an ANN (Artificial Neural Network) module into CDG with the goal of increasing the probability of selecting high coverage test cases. We use assertions coverage as the main metric of coverage rate. The assertions are grouped based on their functional and structural dependencies. Next the randomly generated transactions (i.e., test vectors) are classified using the ANN. Only the transactions with high probability of high coverage will be sent to the DUV. To further enhance the verification process, the assertion groups are prioritized and the priorities of the groups are dynamically changed. Also the low-coverage or redundant assertions will be filtered out. Given the limited verification time, our framework achieves significantly higher assertion coverage than that of the traditional CDG-based frameworks.

The rest of the paper is organized as follows. Section 2 presents the related work on verification acceleration. Section 3 describes the proposed methodology including coverage-directed test generation, assertion grouping, and the ANN setup. Section 4 provides the experimental results on performance improvement of our framework against traditional UVM. Section 5 concludes for this paper.

2 PRIOR WORK

Several research works have utilized machine learning for formal verification based on BDDs [11] (Binary Decision Diagrams [1][2])

and SAT solvers [3][5]. Simulation-based verification has been used as a complement to formal verification, especially in cases that lack detailed understanding of design specifications.

A great amount of effort has been dedicated to improving simulation performance, using simulator optimizations such as GPGPUs, emulators, and sampling. Event-driven gate level simulation, proposed by the authors of [4], was accelerated by general-purpose graphics processing units (GPGPUs). Also a gate level design is leveraged to exploit advantages of low switching activity under large hardware designs. New algorithms for netlist partitioning were developed and optimized for GPGPUs by first doing levelization to make parallization of the netlist easier and then executing one level in parallel only when at least one of its input values have changed, which is event-driven. Finally, the NVIDIA's CUDA architecture was performed to offer a programming interface which could enable users to write software application based on the underlying GPGPUs.

The CDG-based verification proposed in [7], uses Bayesian networks to model coverage information and testbench directives. A dependency graph based on causal relationships between testbench directives (cause) and coverage parameters (effect) is created, which is then trained and evaluated. It can achieve full coverage faster as compared to manual test directive. However, it is more suitable for automatic creation of test directives as compared to human users.

3 OUR VERIFICATION FRAMEWORK

We will explain in this section our ANN-based framework that accelerates the CDG (Coverage Directed Test Generation). We first present the 5 steps of ANN-based CDG. Next we describe how the assertions are grouped according to overlapping control signals, common resources and locality to increase the probability of selecting high coverage test vectors. We also outline the framework setup and phases.

3.1 Coverage-Directed Test Generation

This section proposes a new test generation process to replace manual generation of transactions. Since the relationship between the test transactions and the assertions are non-intuitive, we propose an ANN-based framework that captures the non-intuitive relations and patterns, which ultimately enhances the quality of verification in terms of speed and coverage both. In addition to the relations between test transactions and assertions, there exist patterns among various design bugs (e.g., because designers tend to make similar mistakes in different parts of the design). Also considering the high complexity of the verification flow, AI can be very effective in capturing dependencies and filtering the parameters that turn out to have no impacts on the outcome. We have found ANN as an effective learning methodology for the CDG process. Before elaborating the detail in each step, several terms are needed to be clarified. The transaction represents random generated test vectors under design specs. Based on the previous simulation, bias is generated by scoreboard to control the direction in following simulation. Stimuli are those test vectors have been chosen to drive the DUT in next round. The steps of our ANN-based CDG process are as follows. Also the corresponding flowchart is illustrated in Figure 1.

Step 1: This step is performed to prepare the training data for

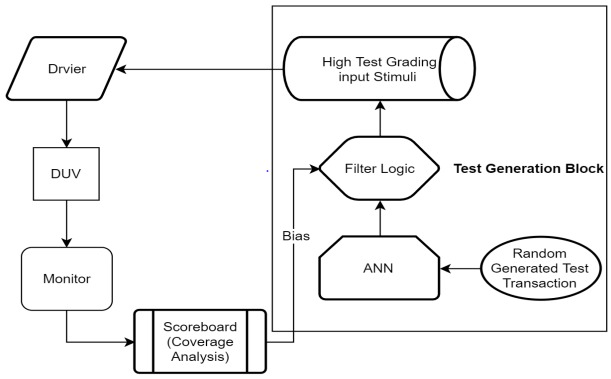

Figure 1: ANN-based CDG process. In the simulation phase, randomly generated transactions will be classified by the ANN. Then, based on the previous simulation, the ANN eliminates irrelevant transactions and remaining transaction set will be treated as stimuli in the next round. This procedure is repeated several times until overall coverage requirement achieved.

the ANN. Randomly generated vectors are sent to DUT. Depending on which assertions they trigger, they will be labeled. Noted that a single test vector is possibly labeled by multiple groups because it tests different aspects of the design. The design properties are written using assertions. As explained in the next section, the assertions are clustered (grouped) using the assertion grouping model.

Step 2: The ANN is trained with labeled transactions generated in the Step 1. Once the ANN is well-trained, it will be embedded as the ANN module into the modified CDG process for the rest of the verification process. Note that to efficiently use the limited verification process, our framework utilizes a conventional CDG, at start while the ANN is being trained, and switches to ANN-based CDG when the training is complete.

Step 3:As shown in the test generation block, the randomly generated test transactions are fed into the trained ANN to predict which assertion group they may target. In case they trigger assertion groups with high priorities (criticality) they will be forwarded to be applied to the DUV. However if the randomly generated transaction is irrelevant, it would be filtered out. One main key to the efficiency of our approach is that our ANN would be significantly faster than the DUV simulator or emulator and hence this steps is a fast process.

Step 4: In this step, the stimuli that was chosen in step 3, is applied to the DUV as shown in the figure 1. The simulation results are monitored. if there exists any assertion group which hasn't been tested, the scoreboard will gives a bias to the filter logic to skew future test case focusing on that assertion group.

Step 5: If overall coverage achieves the test plan expectation, terminate the CDG process, otherwise jump to step 3 with the bias generated in step 4.

Theoretically, in the worst case scenario the number of labels is too large for the ANN to handle, because each assertion can in the worst case be treated as a separate category (i.e., assertion group.) However in practice the number of assertion groups can be very low. This is thanks to the fact that many assertions are related and

therefore own common patterns that can be recognized by the ANN. We will show that only 6 assertion groups are sufficient for a dual-core processor with various tricky bugs, to provide significantly higher verification quality. Our assertion grouping algorithm is discussed next.

3.2 Assertion Grouping

Typically, different assertions may overlap with each other in terms of the design specifications they target. We refer to those assertions as related, and we aim to cluster them into the same group. The reason for multiple assertions to be related, is that same design structures are tested, which means some control signals, execution paths, and arithmetic/memory resources are common. Another reason for related assertions is that the designer tends to make similar mistakes, and then in turn results in assertions having overlaps, in the sense that one failing would imply that the related assertions have a high probability of failing too. Clustering these similar assertions to the same group can significantly increase the efficiency of verification in terms of verification time and coverage. Group assertions has the following benefits: (1) It helps reducing the overall simulation time by shrinking the assertion set. During the functional verification, coverage is a main metric to evaluate the completeness of the test. Each assertion should be triggered by test stimuli enough times to ensure the design meets its specification. If assertion set is too large, enormous time will be consumed to find stimuli which can trigger assertions. Therefore, shrinking the overall assertion set helps engineers avoid redundancy and save efforts and time. (2) In case a certain assertion fails in simulation, design bugs may exist in any components related to this assertion, therefore it is reasonable to put extra efforts on verifying those components. With assertion grouping, since assertions in the same group are interrelated, each time one assertion fails, testing other assertions in the same group can help verify the related resources (datapath, etc.) with high failure probabilities.

In our grouping method, there are three factors contributing to the calculation of correlation, namely number of common structural signals (denoted by $\{f_0\}$), length of the shared datapath ($\{f_1\}$), and common resources ($\{f_2\}$). All 3 factors or a subset of those could be used for assertion grouping. We must however address this concern: Is the process of assertion grouping based on the $\{f_0, f_1, f_2\}$ limited to designs for which the designer knows exactly which structure would relate to which assertion? In other words, if an assertion is written based on a high level behavioral view of the design without clear understanding of how the structure of the corresponding behavior would turn out post-synthesis, would assertion grouping fail? The answer is No. Even for those assertions based on the design behavior, the common control signals can be used to group the assertions. Each factor generates its own grouping results. For the common resources, since the limited number of resources exists in the design, it's acceptable to claim every resource has its own assertion group. The scenario space for the other two factors, however is vast in large, and complex designs, therefore creating one group per scenario is infeasible. We as part of our assertion group method, we propose a clustering technique based on edge-labeled graph modeling to convey the correlation between assertions and efficiently cluster the assertions into groups. Similar ideas are also used in [13] and [10] to reduce the overall assertion size and clustering hardware checker. As shown in Figure 2, the edge-labeled weighted graph is defined as the Assertion Graph $(AG) = \{V, E\}$. The set of vertices, defined as $V = \{v_0, v_1, v_2, \ldots, v_n\}$ represents all assertions required to be inserted for the DUV, with v_n, representing assertion n. The weight of edge $w^i_{a\_b}$ denotes the correlation between two assertion a and b when considering factor i.

To make each edge weight into a dimensionless quantity, $w^i_{a\_b}$ denotes the standard weight between assertions a and b, while considering a factor i. We apply the following normalization function to make the weight scale consistent across different factors:

$$w^i_{a\_b} = \frac{w'^i_{a\_b} - \mu_i}{\sigma_i}$$

$$\mu_i = \frac{1}{N_i} \sum_{ab} w'^i_{a\_b}$$

$$\sigma_i = \sqrt{\frac{\sum_{ab} (w'^i_{a\_b} - \mu_i)^2}{N_i - 1}}$$

where N_i is the number of weights given the factor i, $w'^i_{a\_b}$ is the raw weight between two vertices $\{v_a, v_b\}$ in factor's graph, μ_i is the mean and σ_i is the standard deviation of all edge raw weights for the factor i. After calculating each edge weight, two adjacent nodes with weight higher than threshold δ_{corr} will be grouped together, edge weight lower than δ_{corr} pair will be considered as belonging to two distinct group. All assertions get grouping by three different factors separately. After group merging, the final grouping result will be a assertion group set $G_{assertion} = \{g_1, g_2, g_3, \cdots, g_n\}$ and each individual group contains all related assertion vertices $g_i = \{v_1, v_2, v_3, \cdots, v_n\}$. Note that three factor influence the relation between assertions: the number of overlap control signals f_0, the number of common resources f_1, and the locality which is defined as the length of shared data path f_2. This means in the graph representation, the weight of two adjacent nodes, would be a function of those 3 factors. To illustrate how to calculate the raw weight w'_{a_b} in the Assertion Map, the following provides an example of assertion grouping for a pipeline processor design. The following are five processor assertions used in the example:

$A_1 : ID_{op} == R_{type} \&\& ALU_{src} == ADD \mapsto \#\#[3:4] WB_{out} == ADD_{ref}$

$A_2 : ID_{op} == R_{type} \&\& ALU_{src} == SUB \mapsto \#\#[3:4] WB_{out} == SUB_{ref}$

$A_3 : ID_{op} == M_{type} \&\& MEM_{sl} == SD \mapsto \#\#[2:3] MEM_{wen}$

$A_4 : ID_{op} == M_{type} \&\& MEM_{sl} == LD \mapsto \#\#[2:3] MEM_{en}$

$A_5 : ID_{op} == BEZ \&\& Reg_0 == 0 \mapsto WB_{out} == branch_{taken}$

The following describes each factor and shows how they should be considered towards calculating the raw weights:

3.2.1 Number of Overlapping Control Signals f_0. It aims to capture the similarity between two assertions when considering their control signals. The ratio of the number of shared signals with average number of two assertion signals is defined by the raw weight. For example, for assertion A_3 and A_4, there are one common control signal ($ID_{op} == M_{type}$) and two different signals ($MEM_{sl} == SD$ and $MEM_{sl} == LD$). Therefore the raw weight is 1/2.

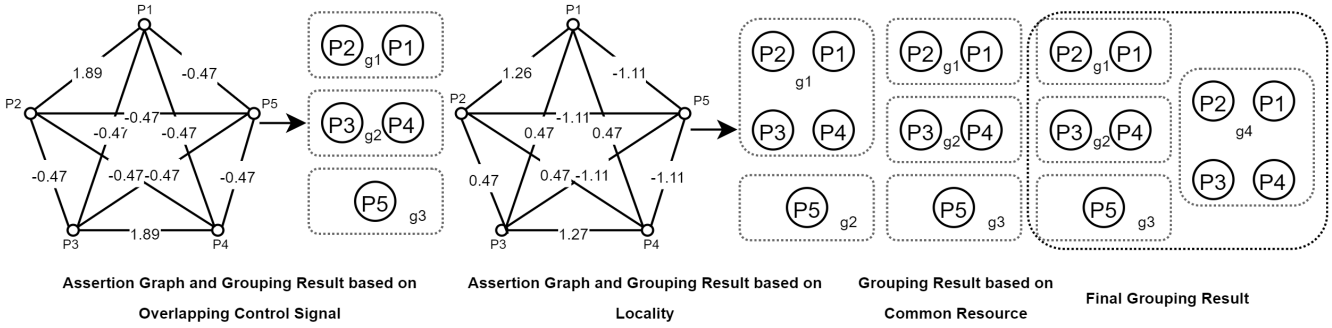

Figure 2: Overview of assertion grouping procedure. First two weighted graphs are constructed based on the structural similarities between assertions. Next, any edge with a weight lower than δ_{corr} is considered as irrelevant, which means the corresponding pair of vertices, i.e., assertions are irrelevant. Finally only the relevant edges remains which means the corresponding pairs of vertices are group. Also the "common resources" would be directly used for grouping. Finally the 3 grouping results are merged into to distrinct groups.

3.2.2 Locality f_1. This factor is used to merge two assertions into a group if their majority execution data paths are overlapped. The ratio of the shared data path with the average individual total data path is calculated as the raw weight for two adjacent assertion vertices. For example, assertion A_1 relates to four stages and A_3 relates to three stages. The common data path between A_1 and A_3 is three stages. Therefore, the edge weight between A_1 and A_3 before normalizing is 3/3.5.

3.2.3 Common Resource f_2. This factor is considered based on the fact that the number of resources for a given design (DUV) is limited, therefore each resource has its own assertion group. Here, A_1 and A_2 are sharing ALU, A_3 and A_4 are sharing memory block and A_5 uses branch predictor exclusively.

As the last step of assertion group algorithm, the groups are checked to see whether any two or more groups perfectly match each other, i.e., they have identical assertions, which means they should merge into one group. Note that groups will have common assertions, therefore groups have overlaps, however merging would occur, if and only if the exact set of assertions appear in two or more groups.

3.3 ANN Setting

Considering the non-trivial relations between the input vectors and the assertion groups, we utilize ANN to identify patterns within transactions. Since the ANN has several layers: the first layer focuses more on the local patterns whereas the following layers are set to capture the patterns from the former layers from higher perspective. The final layer therefore makes decisions from a higher perspective, based on local and more global patterns. This relation between layers and property patterns is advantageous to realize the relationship between input test vectors and the assertion groups. For instance, for the CPU design, the opcode is straightforward and we can anticipate that if certain ALU operations have bugs, then it may relate to certain opcodes. However, the relationship between certain opcodes and whether they will lead to the data dependency problem is not straightforward. Therefore, the ANN can be used to find the correlation in terms of above questions.

The first layer of the ANN is the input layers. The test vectors are the input data. We have examined two configurations: In the first configuration, 32-bit vectors are sent directly to the ANN, which means the input layer has 32 neurons and each input is either 0 or 1. In the second configuration, we categorize the vector bits into opcode, rt, rs, and ALUopcode/offset. Our testing results show that configuration one, i.e., uncategorized vector bits produces more accurate results. This may imply that the patterns could be more too complicated for designer to manually identify, and we should let ANN figure out the patterns automatically. We have therefore chosen configuration 1 to proceed with our experiments.

In terms of the layers and hidden neurons within each layer, we performed several experiments to calculate the optimal configuration. We have chosen a 3-layer ANN including 1 input layer, 1 hidden layer, and 1 output layer with 32, 128, and 128 neurons in each layer, respectively. The number of neurons in the input layer is the same as the number of input bits. The activation function we apply is called ReLU (rectified linear unit) and the training objective function is the cross entropy loss function. It is worth noting that since the input data for our verification problem is not as large as the image or voice data, deep convolutional neural network might not be a proper solution for this problem. It should also be noted that too many layers in the ANN will lead to the lack of data and the insufficienty of the training of the network.

The output layers are the assertion groups. We use the softmax function as the output layer, therefore the output numbers will be fractions, rather than 0 or 1. To classify the vectors into assertion groups, we can either use the probability (i.e., choose the group with the highest probability) or use a threshold for the output value and in case the output value is higher than that threshold, the vector would be associated with the corresponding assertion group.

3.4 Test Phase

After the ANN is well-trained, the ANN will be embedded into the test phase and become a part of the verification flow. Figure 3 shows the whole process of the test phase. During the test phase, the randomly generated transactions will be first sent into the ANN for classification. The ANN will filter the redundant transactions that have no relationship with the existing assertion groups and classify the rest of the transactions to the related assertion group.

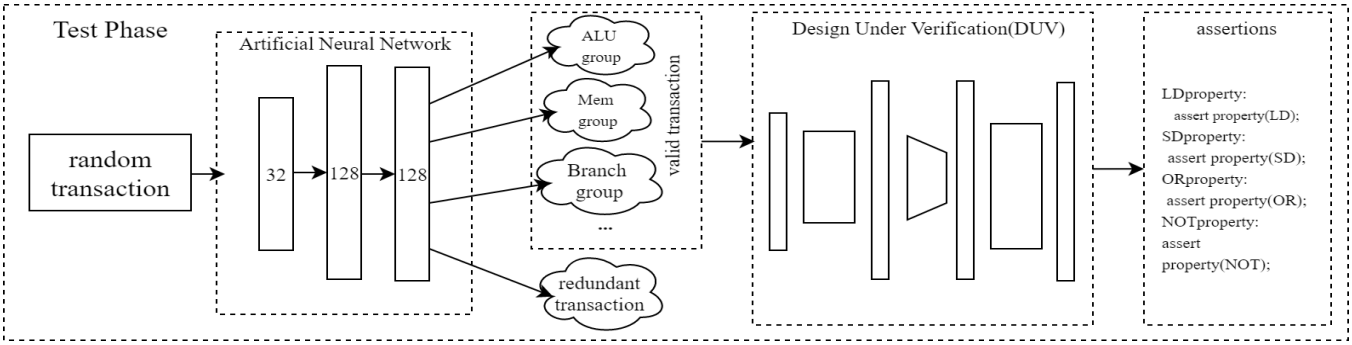

Figure 3: Overview of the test phase. The randomly generated transactions will be first classified by the ANN. The redundant transactions will be filtered whereas the rest of the transactions are valid and will be classified into related assertion groups. After ANN classification, the valid transaction will be sent to the DUV and trigger assertions in order to achieve a higher coverage.

Table 1: Training Time for Different Assertion Groups

	AG0	AG1	AG2	AG3	AG4	AG5
# Neuron	30	30	30	10	30	20
Time (s)	11.3	9.98	13.4	12.1	10.1	11.6

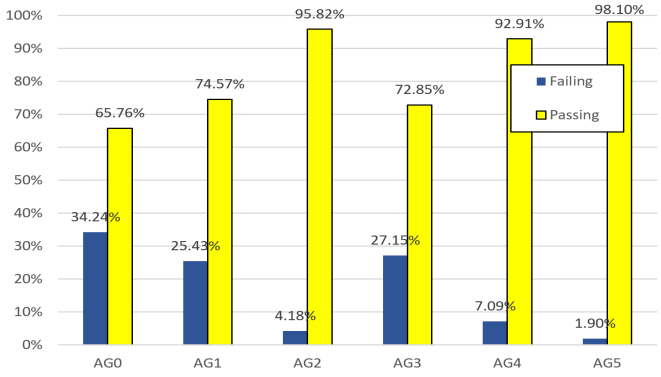

Figure 4: The distribution of passing and failing vectors

The previous assertion groups proved that the assertions within each assertion group are highly correlated. The ANN extracts the relations between the input transactions and the assertion groups. Therefore, the classified transaction will target more on the assertions within assertion groups and have higher probability to trigger the assertions. Therefore, within the limited time, the ANN-based CDG process can achieve a higher functional coverage.

4 EXPERIMENTAL RESULTS

To test the effectiveness of our ANN-based verification framework we have implemented a dual-core RSIC processor, which includes two cores with a static pipeline and DMEM unit, as well as a dispatcher unit and a unified memory. We use the assertion grouping algorithm discussed in Section 3.2 to cluster all processor's assertions into 6 separated groups, i.e., AG0 to AG5. These assertions cover processor specifications such as control hazard handling, arithmetic calculation correctness, memory read/write, etc.

4.1 Configuration and Distribution of Vectors

We chose the Neural Net Pattern Recognition toolbox of MATLAB as our ANN platform. In this experiment, we tested different number of neurons in the hidden layer to calculate the configuration that provides the least error rate and training time for each assertion group. In our experiments, a total of 83620 vectors were randomly generated. The training was based on 70% of those vectors. Also 15% of those vectors were using for testing the ANN during the test phase. Finally we used the remaining 15% for validation, i.e., adjust the labels of some of the vectors, based on the the results of the testing phase. Note that the randomly generated vectors included those that were related to correct instruction set, but also some that result in error due to some structural and behavioral bugs, such as misconnections of control signals or wrong data forwarding etc, that were intentionally added to DUV. The vectors that results in failure were labeled based on which assertion group they trigger. Each label is a 6 digit number $d_5, d_4..d_0$, where $d_i = 1$ means the vector passes all the assertions of group i and di =0 means it triggered at least one assertion from group i. Those labels are stored and used for supervised training of the ANN. Figure 4 illustrates the distribution of passing and failing vectors for each of the 6 assertion groups. Note that the distribution varies among assertion groups, due to the nature of properties related to each assertion group as discussed in section 3.2. Table 1 presents the training time for the 6 ANNs corresponding to the 6 assertion groups, given the number of neurons in the hidden layer. Note that the number of neurons were selected such that accuracy over training time is maximized. As we discussed in section 3.3, the training time of the 6 ANNs corresponding to the 6 assertion groups should overlap in time, to maximize the parallelism. As reported in Table 1, the training time of the ANNs is almost the same, which helps with the parallel training of those ANNs.

4.2 Failing Vector Prediction Rate (FVPR)

There are 4 possibilities for the ANN in terms of how it would treat a given test vector:
Case 1: ANN: Fail, DUV: Fail;
Case 2: ANN: Pass, DUV: Fail;
Case 3: ANN: Fail, DUV: Pass;
Case 4: ANN: Pass, DUV: Pass;

Table 2: Success rate and assertion hit results for the ANNs

Testing Data						
	AG0	AG1	AG2	AG3	AG4	AG5
FVPR	34.36	95.55	86.54	43.06	51.11	71.20
Case 1	1660	2986	463	1518	415	136
Case 1 + Case 2	4831	3125	535	3525	812	191

Out of the 4 cases, only cases 1 and 4 are the ones that are identified by ANN as vectors to be fed into the DUV. This means cases 2 and 3 are referring to vectors that are going to be eliminated by the ANN and would not be applied to the DUV (because ANN calculated them as passing all assertions). Also note that among the 4 cases, case 2 is the one that is wrongly eliminated. Therefore the Failing Vector Prediction Rate (FVPR) of the ANN is formulated as $\frac{Case1}{Case2+Case1}$.

Table 2 shows the Failing Vector Prediction Rate for each assertion group, calculated during the test phase. The number of test vectors during the test phase is 15% of total 83620, i.e., 12543. Note that FVPR is calculated based on the ratio of row 2 in Table 2 divided by its row 3. Also note that FVPR for each ANN is independent of the FVPRs of the other ANNs. Also note that all the 6 ANNs are used in parallel to decide whether a future vector should be applied to DUV or dropped. A vector is dropped only if all the 6 ANNs report it as Pass, i.e., case 3 or case 4 vector. This implies that the average FVPR among all the 6 ANNs would be a measure of the quality of verification in terms of managing the randomly generated vectors, i.e., dropping them, or applying them to DUV.

4.3 Performance Evaluation

In addition to using FVPR to make comparisons among different ANN-based implementations of our framework, there are two metrics to evaluate the performance of our ANN-based framework when compared to a baseline technique: one is the the quality of verification in terms of vectors triggering assertions in a limited time. This could be looked at as a measure of coverage, because the higher the number of triggered assertion, the higher is the number of detected design bugs. We refer to this metric as assertion coverage. Another measure of performance is the time it takes to process a given set of test vectors. First we compared our ANN-based CDG verification process with the traditional CDG assuming both given a fixed verification time limit. This comparision would help us measure the impact of ANN training time overhead. As shown in the Table 3, given the same time, our ANN-based CDG determined 1660 vectors for AG0, as case 1 vector (as defined in section 4.2), which means those vectors were predicted to trigger at least one assertion in AG0, and they indeed triggered at least one, when applied to DUV. However using the traditional CDG, only 380 were able to trigger an assertion from AG0. This shows a minimum assertion coverage improvement of 4.37x considering a limited verification time. Note that the assertion coverage improvement is as high as 28.94x. Next we considered the same number of randomly generated vectors (in this case about 12000 vectors) and compared our framework with the traditional CDG in terms of the verification time it takes to process those vectors. The traditional CDG took 1.84 seconds, wherease our ANN-based CDG took a maximum of

Table 3: FVPR comparison between our ANN-based and the traditional CDG.

	AG0	AG1	AG2	AG3	AG4	AG5
Traditional CDG	380	201	16	271	89	8
ANN-based CDG	1660	2986	463	1518	415	136
Improvement	4.37	14.86	28.94	5.60	4.66	17

Table 4: Simulation time comparison between our ANN-based and the traditional CDG.

	AG0	AG1	AG2	AG3	AG4	AG5
Traditional CDG	1.84	1.84	1.84	1.84	1.84	1.84
ANN-based CDG	0.075	0.0747	0.073	0.0534	0.0672	0.0586
Speedup	24.53	24.63	25.20	34.46	27.38	31.40

0.075, which is 24.5 times as fast as the traditional CDG. The reason for this significant verification speedup is that our ANN-based framework can label the vectors in a very short time, must faster than the simulation on DUV, and drop the case 3 and case 4 vectors.

5 CONCLUSIONS

Design complexity of modern processing systems has significantly impacted the reliability of conventional test generation. Our framework integrates ANN into the CDG (Coverage-Directed test Generation) process to accelerate the verification process and increase the coverage in a limited verification time. The assertions are clustered using an assertion grouping algorithm with the goal of efficiently managing properties and the relations among them. The ANN is used to label the randomly generated vector based on which assertion group it would be associated with. If test vectors trigger assertion groups with high priorities, they will be applied to the DUV. However if the randomly generated transaction is irrelevant, it would be eliminated. Our experimental results confirm that our ANN-framework can eliminate as high as 40.2% of test vectors with much faster runtimes, when compared to a standard SystemVerilog testbench implemented in UVM. The resulting time speedup reaches 24.5x and assertion coverage improvement ranges from 4.37x to 28.94x.

REFERENCES

[1] R. E. Bryant. "Graph-based algorithms for boolean function manipulation". In: *TC* (1986).
[2] R. E. Bryant. "Symbolic Boolean manipulation with ordered binary-decision diagrams". In: *CSUR* (1992).
[3] B. Bünz et al. "Graph Neural Networks and Boolean Satisfiability". In: (2017).
[4] D. Chatterjee et al. "Event-driven gate-level simulation with GPGPUs". In: *DAC*. 2009.
[5] D. Devlin et al. "Satisfiability as a classification problem". In: *AICS*. 2008.
[6] D. Ernst. "Complexity and internationalisation of innovation". In: *IJIM* (2005).
[7] S. Fine and A. Ziv. "Coverage directed test generation for functional verification using bayesian networks". In: *DAC*. 2003.
[8] O. Guzey and L. Wang. "Coverage-directed test generation through automatic constraint extraction". In: *HLVDT*. 2007.
[9] C. Ioannides and K. I Eder. "Coverage-directed test generation automated by machine learning–a review". In: *TODAES* (2012).
[10] M. H. Neishaburi and Z. Zilic. "Enabling efficient post-silicon debug by clustering of hardware-assertions". In: *DATE*. 2010.
[11] M. L. Rebaiaia et al. "A neural network algorithm for hardware-software verification". In: *ICECS*. 2003.
[12] R. R. Schaller. "Moore's law: past, present and future". In: *IEEE spectrum* (1997).
[13] J. G Tong et al. "Assertion clustering for compacted test sequence generation". In: *ISQED*. 2012.