Machine Learning Based Verification Planning Methodology Using Design and Verification Data

Hanna Jang[1] Seonghee Yim [2] Sunchang Choi[3] Seonil Brian Choi[4]
[1]Samsung Electronics, Hwaseong, Korea.
[1]hn01.jang; [2]sh.yim; [3]s86.choi; [4]seonilb.choi@samsung.com

***Abstract*- As complexity and integration of SOC continue to increase, its development time increases significantly and the verification time does increase even more dramatically. In a typical verification process, a large number of directed or randomized simulation test cases are generated and run as a regression form. In a full chip verification environment, this regression takes significant amount of time to be completed and could be a critical path in a project schedule. In this paper, we discuss the methodology to efficiently manage the regression and reduce its run time. We classify test cases in regression by using various Machine Learning algorithms and correlate with the amount of design changes and previously failed simulation results. During this process, we are able to predict failing test cases in the early regression stage otherwise could find them in the later regression stage. It gives an additional debugging time and eventually help us to reduce total regression time. The proposed methodology is being applied to the latest mobile SOC projects.**

I. Introduction

As mobile smart devices increasingly provides more features, SOC (System On a Chip) design size and complexity are also increasing in a very rapid pace. In particular, its verification complexity and time increases exponentially white the development time for SOC rather shortens. This means that it is becoming increasingly difficult to satisfy the TAT on time. To overcome these challenges, this paper proposes an optimal verification planning methodology and strategy based on data analysis and machine learning.

An SOC product is designed based on project specifications and then full-chip level scenario-based design verification and Block /IP level functional verification are performed for multiple development stages. As a SOC project progresses, design changes continue to occur at each development stage, and two main types of design changes exist from the verification perspective: (1) increasing design maturity and (2) increasing design instability. To meet the overall development schedule and maintain the design maturity, the completeness of verification in each development stage is a very serious matter.

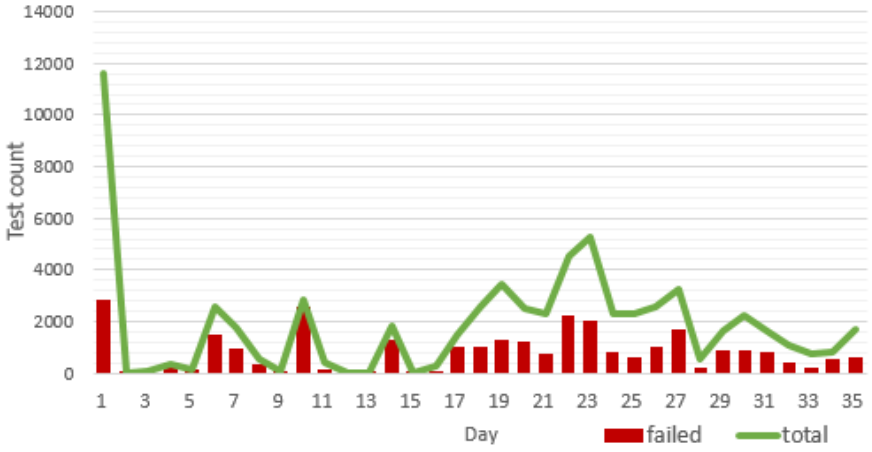


Figure 1: The Amount of Total Tests and Failing Tests during Regression

During the course of the project, RTL is changed and released multiple times as results of design changes. Once new RTL is released, the person responsible for the verification should complete the regression tests for verification as soon as possible to prove the integrity of the RTL. Since many errors may occur and many debugging might be

required during regression tests, it would be better to verify the cases with a high probability of failure first and debug them reliably related to the RTL bugs or other environmental bugs. However unfortunately, a huge variety of tests included in regression process for verification are performed randomly without any priority. Because it is very difficult to predict which test will fail before the regression test is completed [1]. For this reason, the verification engineers are facing new failures almost every day during the entire verification process. As shown in Figure 1, a new failure occurs every day during about 35 days of regression test for verification. The green line means the total number of verified tests on that day, and the red bar shows the number of failed tests on that day.

The failures are found just before the verification target date not only puts the verification owner in trouble, but also makes the verification quality worse due to the short debugging time. Therefore, finding tests that will fail and verifying them first at the beginning of the verification period is one of the most important factors in reducing the verification period and enhancing the verification quality. In order to find tests includes failure cases at each development stage, the test case should be analyzed and classified by verification scenario, and the correlation between the failure occurrence rate and the amount of design change at each development stage also should be analyzed.

In this paper, the verification scenarios(full-chip level scenario-based design verification test cases) are clustered and applied to the deep learning model. And we analyze the correlation between verification scenario, design domain (ex. design hierarchy, power, performance and specification) and development stage to confirm if there is actually a meaningful relationship.

II. BACKGROUNDS

Machine Learning: Clustering & Classification

Machine learning (ML) is widely used to predict what input data means. Whether the predicted result is right or wrong can be checked by comparing with the actual answer. Machine learning methods can be divided into supervised learning and unsupervised learning according to the presence or absence of correct answers. In this study, **verification scenario**, which will be applied as input to ML models, is too large and various to be labeled directly by humans, so potential labels were obtained through clustering using k-means, and then ML models were learned using those labels from clustering process as answers.

A clustering is a technique of grouping data into several clusters and finding groups of similar clusters. Thus, objects in one group are more relevant than objects in another group. The **k-means** are one of the clustering techniques, it divides input data into k clusters with high similarity or high relationship depending on the potentially discovered features. Recently, a variety of ML libraries have been provided, so it is not difficult to use k-means, but it is entirely up to the user to find the K value that controls the clustering results. Fortunately, various methods of obtaining K have already been studied, and Elbow and Silhouette, which are the most commonly used, were used in this paper. **Elbow** is the most commonly used method, using Within Cluster Sum of Square (WCSS), which explains the **total volatility** within the cluster, and a cluster with a low score is considered the most suitable K value. **Silhouette** is a technique that measures **how similar** an object is to its cluster compared to other clusters. The score ranges from -1 to 1, and clusters with high scores is considered the most suitable K value.

After obtaining a potential label through clustering, the label is used to learn the classification model. The answer given by the model is called a class. In this paper, we implemented a deep learning model using Pytorch in python library for convenience of GPU use and flexible code application. The label obtained from the k-means processes and verification scenario were used as input data, and **the train set** and **test set** consisted of **8:2**, and finally **the performance of 93% correct answer**.

Machine Learning: Dimensionality Reduction & PCA

There is a phrase called the curse of the dimension. When dealing with ML, the number of features that can specify the object of analysis is called dimension. As the dimension increases, the possibility of correlation between each feature increases. Therefore, noise between features increases. In addition to the high dimension of data having a lot of noise, the computational complexity required for model learning and reasoning increases exponentially, and the number of necessary data increases in order to distinguish each dimension. To reduce dimensions, experts can analyze data to select unique features or create new features by technology of **reducing dimensions**. **Principal component analysis** is one of the technology.

Principal Component Analysis (PCA) is a statistical approach that is based on orthogonal transformation to transform possibly correlated features into a set of linearly uncorrelated features known as principle components [3]. After the transformation through PCA, the first component has the largest possible variance and the second

component has the second largest variance under the constraint that is orthogonal to the first component and so on [4]. PCA results in newly extracted vectors that are an uncorrelated orthogonal basis set. It is important to note that PCA is sensitive to the scaling of the original features. In this study, we implemented the PCA algorithm using the Scikit-learn library of the Python software package [5].

III. ARCHITECTURE

In this chapter, we describe the architecture of our data management and analysis system that collects design and verification data, and builds ML models.

SOC Design and Verification Flow

The design and verification of the SOC hardware consists of four processes. The first process is to define SOC specification. It is followed by IP development, block development, and then SOC integration processes. IP development carries out in accordance with SOCs development schedule. As IP development begins, the verification of the IP starts, in addition, a block, where a block is a set of IPs or a sub-system, is built. The block design and verification include inter-IP connections within the block. In parallel, connecting those blocks starts at SOC or full chip level.

In this paper we define verification steps for each development stage and defines them as Verification Level (VL). In VL1, IP level design and verification are performed. Block level design and verification, which is verification for a set of IPs or sub-systems, follows. In VL2, IP level design and verification as well as block level design and verification are mainly performed. Design and verification for full chip SOC design start. IP level design and verification are complete before VL3. In VL3 and VL4, the design and verification for the blocks and full chip SOC are mainly performed. In VL4, a regression, which is batching a large number of test cases, for blocks and SOC is performed mainly [2].

Verification Scenario

In design/verification process of SOC product, test case and test specification of verification are different depending on specification and development stage. Verification of each development stage includes verification of **full chip level functions** as well as of each **Block/IP level functions**. Also, the verification scenario includes the sequence information of the function and the register setting information of the related design.

Design Domain

During design and verification of SOC, various types of data exists. We categorize our design in 25 specification areas such as bus interface, register, ports, etc., which is called **a metadata design domain**. In order to analyze various types of design and verification data, we collect three types of data. The first type is verification data (e.g. verification code changes), the second is metadata of design (e.g. IP-XACT components), and the third is design data (e.g. RTL changes). We collect the changed information for those 25 representative metadata for specification.

Design Data Analytics (DDA) & Design Verification Automation system (DVA)

Figure 2 shows the system that is configured to apply the experimental flow to each development stage of SOC projects. During an entire SOC development, design and verification data are released almost every week with a new name called label from the design repository. Design data includes registers, integration information, and RTL of its own. The verification data during the verification process are also changed almost every week. In this development stages, numerous data are generated, deleted, and changed. To manage such a large quantity of these data, we have developed a system called DDA, which collects and analyzes those changed data in each design domain. Figure 2 shows three modules of the data types supported by DDA, although more types of data are generated and covered in the SOC development process.

We have also developed another system called DVA, that is originally used for collecting verification scenario data and managing regression runs at each development stage. In particular, DVA uses the regression manager to collect all verification data (e.g., LSF information, run results, run logs, environment). We add the verification & triage scenario planner based on the ML cluster & classification model to the original DVA system. Changes data in each design domain analyzed by DDA and verification scenario data analyzed by DVA are used as the training data of the ML model. In this paper, 1.7 million of simulation results from verification scenarios through DVR and 4.2 million design files through DDA were used.

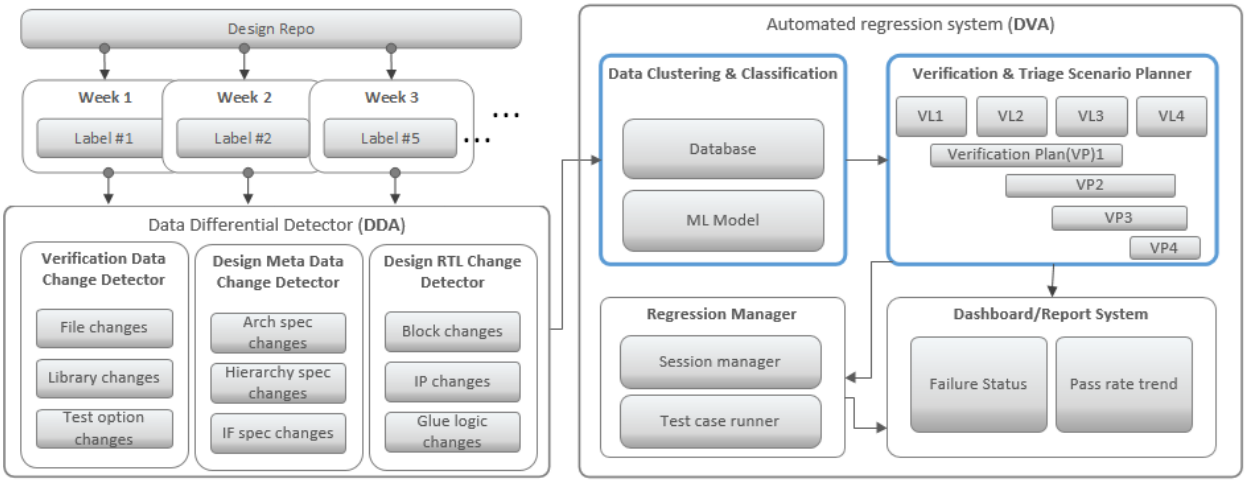


Figure 2: Overall Architecture or Experimental System

IV. Verification Planning Methodology

Changes in RTL as a target of verification are bound to affect the verification scenarios, but it is not common to analyze the correlation between design and verification because the design and the verification are usually conducted by separate organizations. However, we analyzed the correlation based on the following three assumptions.

(1) As mentioned in background above, a verification scenario includes various design information and simulation conditions or options. We classify a certain verification scenario based on these options. (2) Any RTL changes would affect test results. 3) By analyzing the correlation between (1) and (2), we re-order the dispatching priority of the verification scenarios during a regression run at each development stage, which would reduce total regression run time or increase verification completeness quicker.

ML Based Scenario Clustering & Classification

In order to analyze the relationship between the verification scenarios and the design domains, it is necessary to identify the characteristics of each verification scenario and then cluster the verification scenarios according to the characteristics. The data prepared for clustering includes millions of verification scenarios obtained from five SOC projects. The best quality clustering method would be for a person to identify the composition of verification scenario and specify what characteristics it has. However, because the amount of verification scenarios is too large and various to be classified manually, clustering through K-means is performed.

When analyzing the regression data, the first challenge we faced is that the number of features to analyze is too large. Since reducing the number of features and dimension is essential to achieve high performance of ML models, we used Sklearn's KernelPCA to reduce the dimension for string as nonlinear data. By checking the cumulative contribution rate, the feature count is reduced to hundreds.

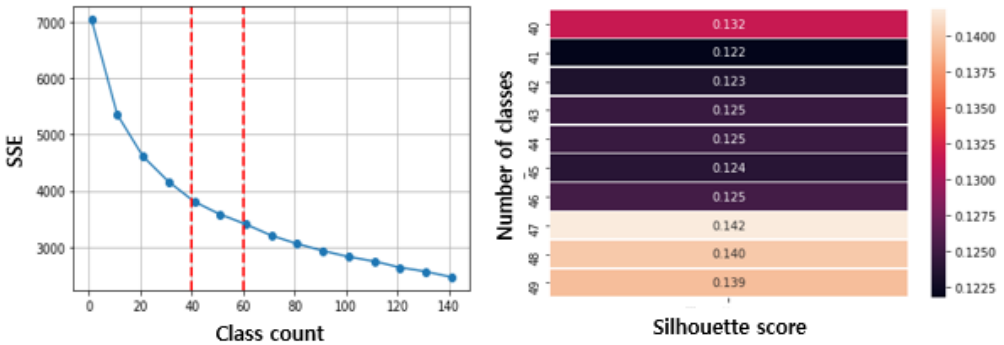


Figure 3: The Experimental Result of Elbow and Silhouette Method.

The number of clusters to classify our verification scenarios is determined based on the experimental results of Elbow and Silhouette method. In Figure 3, the result of the Elbow show that the larger number of classes, the smaller the value of SSE (Sum Square Error). However, when the model is actually trained with more than 60 classes, its performance tended to be rather poor. Therefore, the range of the number of class is fixed from 40 to 60 through Elbow, and the experiment is additionally performed with Silhouette method. Finally, it is determined that the number of clusters for the deep learning model is 47(For projects A and B, only 44 classes are valid).

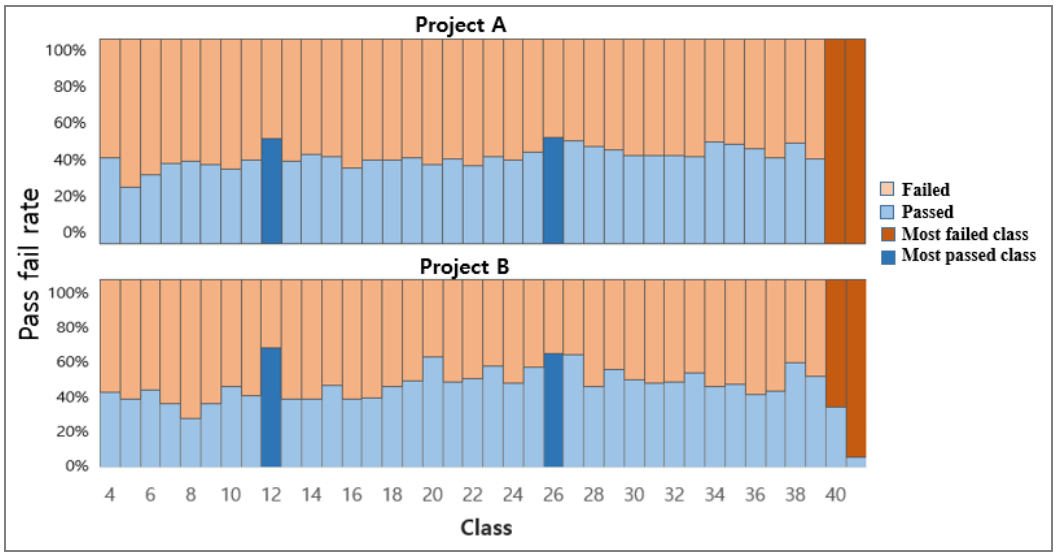


Figure 4: Comparison Result of Pass/Fail Case classified by ML Model

Figure 4 is the result of classification into a model that is trained with the same regression for two SOC projects (Project A and B). Since the regression run generally starts at the VL3 stage, the VL3 results of two projects are compared to ensure that the verification scenarios for the regression is well classified. The results of clustering and classification are considered reliable because the trends of each class of different projects are similar. The subsequent process is analyzed based on Project A since the verification scenario data of all VL steps existed.

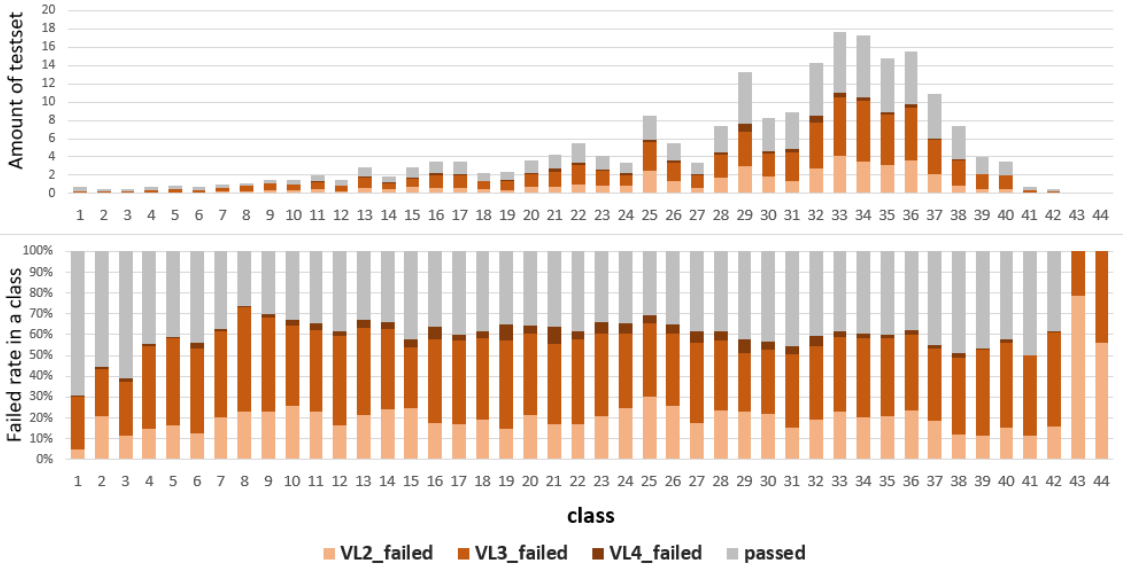


Figure 5: Failure Rate with Verification Stage and Class

Analyzing Verification Scenario Failure with Development Stage:

By classifying the verification scenarios of all development step of the SOC project through the model prepared above, the results in Figure 5 can be obtained. It shows the failure rate for each class and development stage. For example, more failures occurred in class 2, 43, 44 in VL2, class 1, 38, 41 in VL3, and class 19, 21 and 29 in VL4, respectively. If the class is checked early in each development stage, failures can be found more quickly.

Analysis of Design Domain Changes by Development Stage

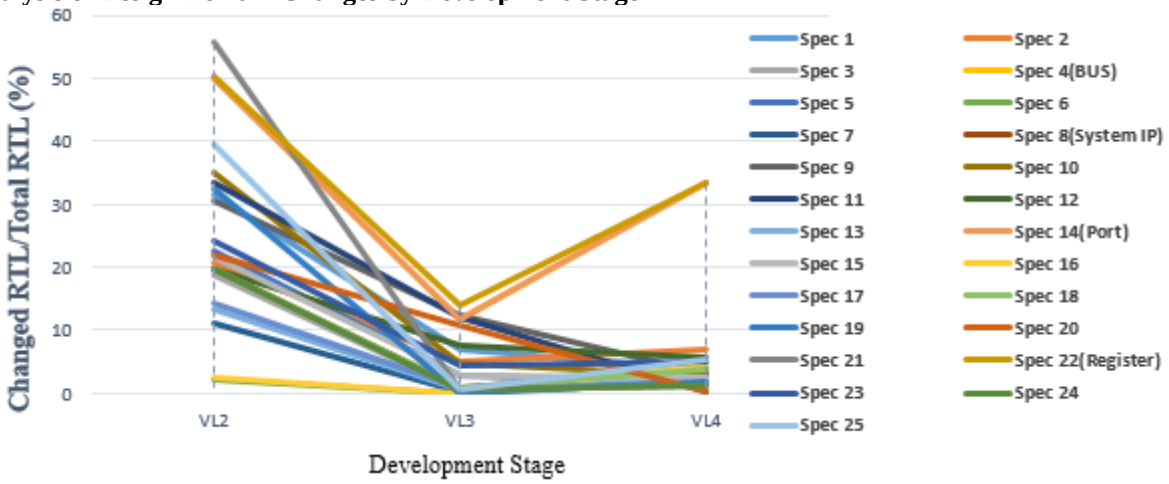


Figure 6: Changed RTL/ Total RTL with Design domain and stage

As shown in Figure 6, the RTL change trend according to the VL stage differs for each design domain. Most specifications tend to converge from VL2 to VL4, but some specifications tend to differ. In addition, some specifications are stable with little change from VL2, which is an early stage of development. What stands out is that in the case of Spec 4, 8, 14, 16, 22, and 25 (such as bus interface, port, register), it can be seen that the amount of changes is higher in VL4 than in VL3. In this way, each specification shows its own trend in the development stage. It provides a guide to check whether such trend affects the regression run result.

Correlation between Design Domain Changes and Verification Scenario

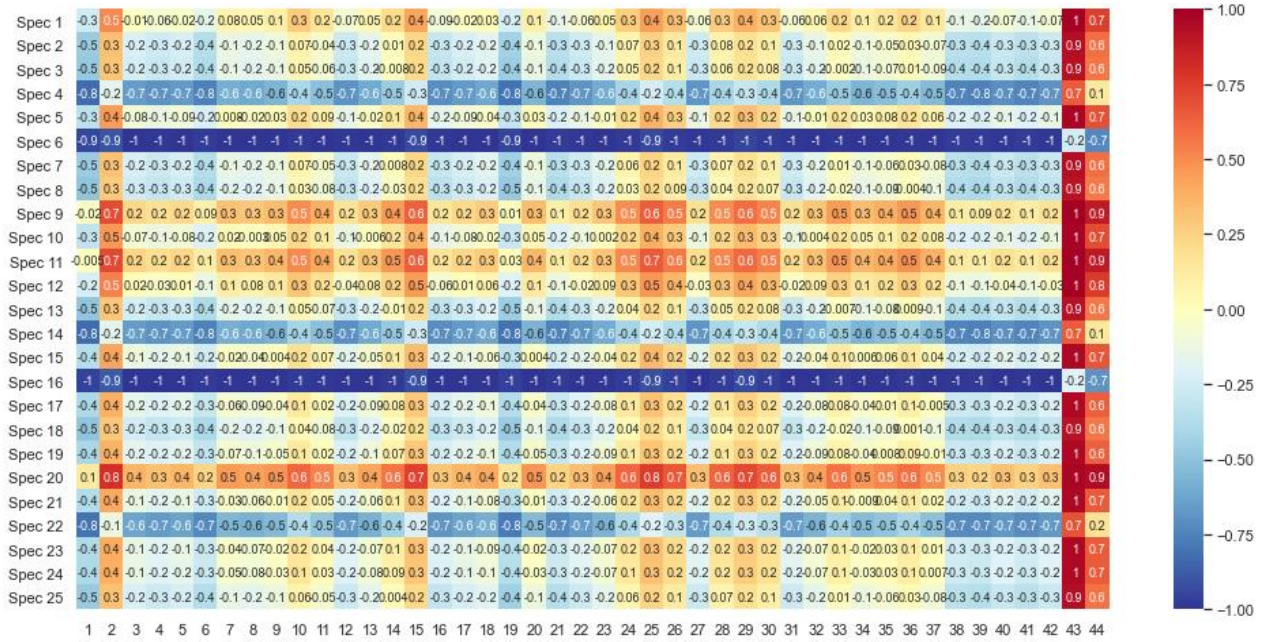


Figure 7: Correlation between Design Specification changes and Verification Scenario

Based on the results of the previous steps, the correlation between changes by design domain and verification scenarios is analyzed. As Figure 7 shows, changes in Spec 9, Spec 19, and Spec 21 (such as component instance, register) cause failure in almost all classes. Especially class 2, 43, 44 that had many failures in VL2 stage, are sensitive to almost all design domain changes. Note that the amount of change for Spec 4, 6, and 16 (such as bus and port) is inversely correlated with failure and thus this specific change rather affected the direction of stabilizing RTL. Using this correlation table, we choose the job dispatch ordering of verification scenarios in regression.

V. RESULTS

The experimental environment used in this study has been applied to the flagship mobile SOC and the efficiency can be measured from the full chip SOC level applications. Based on the data from verification regression for each development stage, a scoring by considering failures and design changes of each class were used and the re-ordered regression schedule with a specific class based on those score are shown in Figure 8.

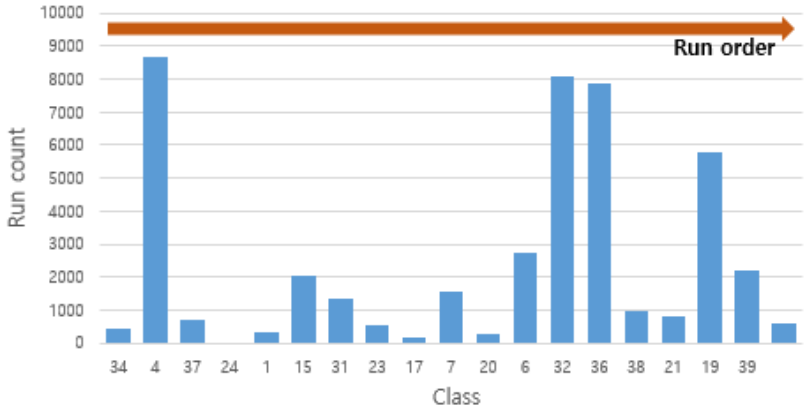
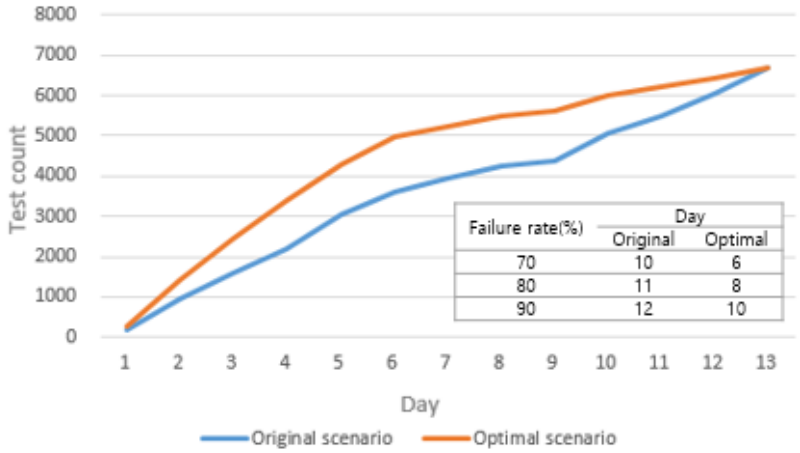


Figure 8. Re-ordered Regression Schedule with Classes



Failure rate(%)	Day	
	Original	Optimal
70	10	6
80	11	8
90	12	10

Figure 9. Comparison of Failures between Real and Optimized Verification Scenario.

Based on the scoring results, the verification schedule is optimized and compared with the existing verification results. If the verification plan for each class is optimized, the failure cases occur early, and thus the debugging time is reduced and the pass rate can be raised quickly. Figure 9 shows that the verification time for each failure rate is 70% : 6 days (optimal) vs. 10 days (original), 80% : 8 days (optimal) vs. 11 days (original), 90% : 10 days (optimal) vs. 12 days (origianl). The result is a verification stability that can be secured only by changing the order of execution of the case based on the scores. With the verification period for each label in the VL3 stage from 13 to 15 days, rapid securing of debugging time and pass rate of 4 to 2 days can improve the progress rate of verification functional pass rate by 21.5% up to 80% of VL3, which is 64% of the total verification period.

It is confirmed that the debugging time, that is, the switching period from class to pass, which has a critical effect on the test optimized in actual VL3, takes 1.3 days for the median value, 1.9 days for the average value, and 14 days for each debugging period occurs.

VI. CONCLUSIONS

In this study, we have analyzed the correlation between design changes and verification scenarios using machine learning. Based on this analysis we developed a method to optimize the schedule of verification jobs or a batch of test cases. This helps us to re-order the set of verification scenarios during the regression, run would-be-failing verification scenarios earlier than others, and eventually save our verification time.

To achieve this, we train and build our ML model using the verification scenarios of 5 SOC projects, failure patterns from 2 SOCs, and design changes in each development stages from one generation old SOC. Using this model, the verification job scheduling is optimized for the current SOC project. We have several interesting observations. Throughout all SOC projects, every design domain has similar failing patterns for a given development stage. This means that the failing cases and their failing time could be predicted for a chosen design domain, which would help prepare our verification planning in advance. Another observation is that the changes of some design domains cause more failing test cases while others decrease failing test cases. For example, the changes on the design domains such as bus interfaces or ports tend to made more test cases passing while the changes on the design domains such as registers tend to cause more test cases failing. This also gives some hint to prepare our verification planning for given design changes.

Our analysis methodology and system still have several areas to be improved. It is required to collect and preserve the data at all development stages of all or major SOC project to build better ML model. To refine our ML model to predict our verification status more accurate, code and functional coverage information would be required to incorporate in our analysis. This might give more precise root-cause of failing test cases. This would greatly reduce our debug time and the total verification time.

References:

[1] Ahmed Wahba, Justin Hohnerlein, Farhan Rahman, "Expediting Design Bug Discovery in Regressions of x86 Processors Using Machine Learning", Advanced Micro Devices, Inc. (AMD), 2021

[2] Seonghee Yim, Hanna Jang, Sunchang Choi and Seonil Brian Choi, "Accelerating SOC Verification using Process Automation and Integration" Design Verification Conference (DVCon), 2020.

[3] Performance Analysis of PCA, Sparse PCA, Kernel PCA and Incremental PCA Algorithms for Heart Failure Prediction

[4] K. Kim, "Face recognition using principle component analysis", International Conference on Computer Vision and Pattern Recognition, vol. 586, pp. 591, 1996. (20)

[5] S. J. Zhao, J. Zhang and Y. M. Xu, "Monitoring of processes with multiple operating modes through multiple principle component analysis models", Industrial & engineering chemistry research, vol. 43, no. 22, pp. 7025-7035, 2004(21)