A Platform-Based Approach to Reduce Verification Turn-Around Time in Memory Designs

Junho An[1], Seonyong Lee[2], Minseon Kim[3], Jaehyun Park[4], Gyehyun Na[5], Dongeun Lee[6], Kiuk Kim[7]
Samsung Electronics, Hwaseong-si, Gyeonggi-do, Korea
{jh0119.an[1], sj91089.lee[2], ms7489.kim[3], gh.na[5], dongeun1.lee[6]}@samsung.com
Synopsys, Sunnyvale, US
{jaehyunp[4], kiuk[7]}@synopsys.com

***Abstract*- Large-scale regression testing in memory verification requires extensive manual effort for log analysis and debug reruns. In conventional distributed environments, regression monitoring, log analysis, waveform debugging, and coverage analysis are executed across multiple tools, leading to inefficiency and prolonged turnaround time (TAT). This paper presents an integrated verification platform built on a commercial EDA tool to streamline these fragmented processes. The proposed platform unifies log analysis, automatic debug rerun, waveform correlation, and coverage analysis within a single environment. Using a modularized checker structure, it automatically extracts fail keywords, instance locations, and timestamps from simulation logs, generates fail buckets, and links them to corresponding waveforms through the Regression Debug Automation (RDA) feature. In particular, due to the deep hierarchical structures of high-density memories, correlating simulation logs with waveforms enables efficient parsing of relevant debug signals. In experiments with 1,500 testbenches, the proposed platform reduced manual debugging period by 50%, decreased regression iterations from three to two, and shortened total debug TAT by 33%. These results have shown a reproducible and scalable methodology that significantly enhances debugging productivity and workflow efficiency in large-scale memory verification.**

I. Introduction

With the advancement of memory technology, various functions and options are now supported, and verification complexity has increased exponentially. To implement diverse application products such as SSD and microSD card using NAND Flash memory, circuit operation must be guaranteed for each combination of functions. Consequently, circuit design has become more complex and the number of verification cases has surged. As circuit complexity increases, the likelihood of errors also rises, and the time required to debug simulation results becomes a major factor delaying the overall verification turn-around time (TAT) [1]. Since debugging consumes significant engineering resources, reducing debugging time is essential for improving verification efficiency. This paper proposes an integrated verification platform technology to enhance memory verification efficiency and discusses its effectiveness.

II. Previous Work

In conventional memory verification environments, individual tools executed on a Linux terminal were used to perform regression monitoring, log analysis, debug rerun, circuit debugging, and coverage analysis. However, this approach required frequent switching between tools, which reduced efficiency and limited functional integration. In particular, when many failures occurred in large-scale regression tests, engineers had to manually inspect logs to classify failure causes. Even during waveform debugging, engineers had to manually trace back from the final failure point to the root cause. This method slowed analysis and prolonged debugging time due to the lack of linkage between fail logs and waveforms [2]. In addition, when using in-house tools, it was difficult to apply the latest debugging methods provided by commercial EDA tools. These limitations constrain the maximization of verification efficiency.

III. Proposed Integrated Verification Platform for Memory Designs

This paper proposes a platform-based approach focusing on five key improvements to enhance verification efficiency by addressing the limitations of conventional methods.

A. Unified Platform Environment

In the conventional distributed verification flow, as shown in Fig. 1, regression monitoring, log analysis, debug reruns, waveform debugging, and coverage analysis were performed through separate tools. This fragmented approach caused efficiency loss due to frequent tool switching and lack of connectivity between functions. In addition, internally developed scripts or tools required dedicated maintenance engineers and limited access to advanced debugging features provided by commercial EDA solutions.

To address these issues, a single integrated verification platform has been implemented using commercial solution, SNPS Verification Management System (VMS), as shown in Fig. 2. The platform integrates regression monitoring, log analysis, automatic debug re-runs, waveform debugging, and coverage analysis into a single environment. By eliminating tool-switching overhead and unifying the verification workflow, the solution has improved both operational efficiency and maintainability.

B. Log Classification Optimization Using Modularized Checkers

1) Comparison Between the Conventional and Proposed Approaches: In the conventional flow, as shown in Fig. 3, fail classification was performed manually by inspecting logs. In large-scale regression environments, waveform dump files were not generated for every test. However, when a failure occurred, engineers had to perform individual debug runs to generate waveforms for analysis. This process increased storage usage and introduced repetitive manual work for identifying root causes. After debugging failures in basic functions such as power-up, erase, program, and read, engineers had to re-execute regressions (Loop-A) to verify that no residual failures remained. This iterative process was repeated until all failures were resolved. In contrast, the proposed method, shown in Fig. 4, automatically classifies memory test results using 136 predefined fail keywords to generate error buckets. Each bucket automatically selects a representative testbench and executes an automated debug re-run. This approach reduces storage overhead and automates both fail classification and debugging based on simulation logs. Automatic classification is applied to both basic and extended functions, eliminating the need for repeated regressions (Loop-A). Because failures are organized by detailed function, engineers can focus directly on waveform debugging of the relevant function. As a result, fail causes are automatically segmented, and the combination of commercial EDA error bucketing and auto-rerun capabilities has minimized repetitive manual work in large regression environments.

2) Modularized Checkers for Log Classification: To enable efficient log classification, the verification environment includes three modularized checkers that operate in a complementary manner. As shown in Fig. 5, these checkers capture memory sequences and Pass/Fail information, print fail keywords to the log, and automatically categorize 136 memory functions, recording the corresponding function name whenever a failure occurs to enable automatic bucket generation. Even if multiple functions share the same root cause, each function's failure is logged separately. To fully eliminate residual or variant root causes in basic functions, the system automatically re-runs the representative testbench of each bucket, allowing the verification loop in Fig. 3 (Loop-A) to achieve near-complete fail coverage with only one iteration. In addition, when a failure occurs in the memory DUT core operation or in a Peri module, the checkers log predefined fail keywords along with the corresponding memory core or peri instance information. Because instance-level data is embedded in these logs, engineers can quickly trace deep hierarchical protocol stack failures to the exact instance level, which greatly reduces debug navigation time and has improved fail-cause analysis efficiency in large-scale regression environments. As shown in Fig. 6, Fail keywords are predefined from the verification plan, detected by the modularized checkers during execution, and refined through regression-based tuning when finer classification is required.

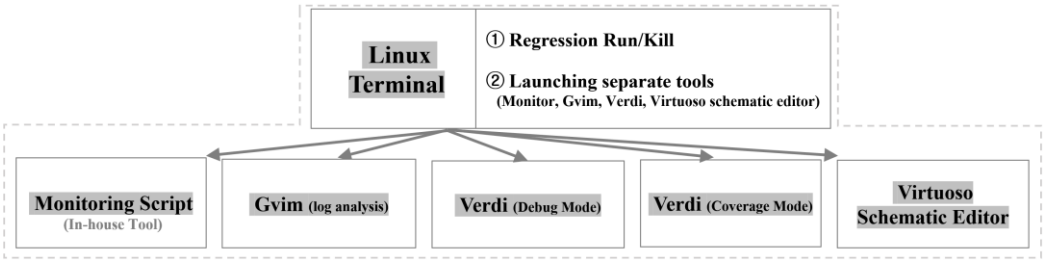

Figure 1. Conventional distributed verification methods

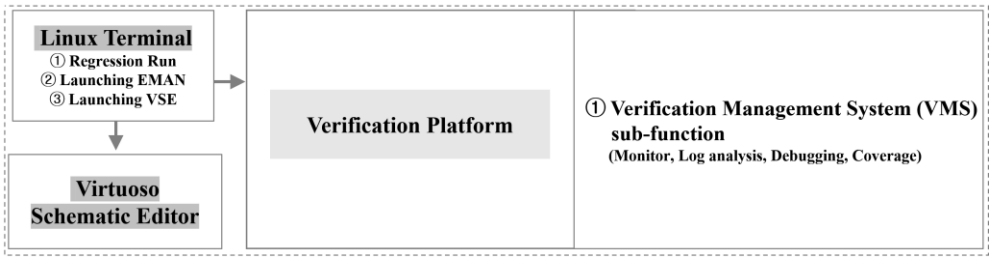

Figure 2. Proposed integrated verification platform

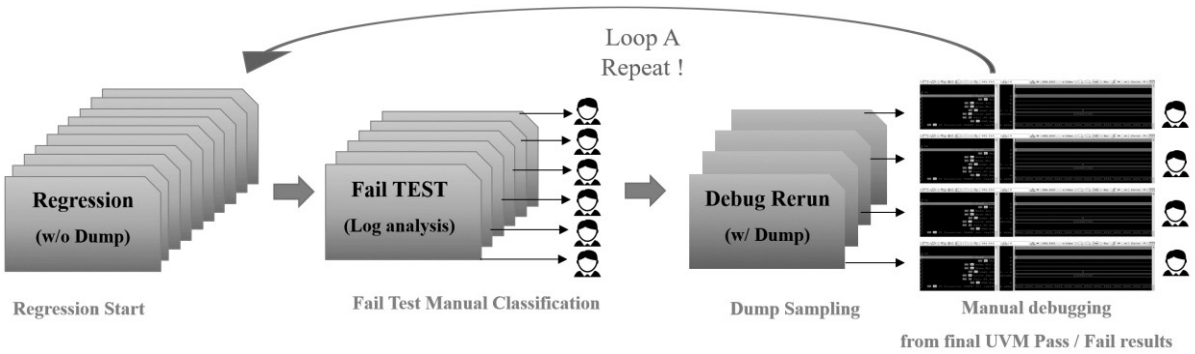

Figure 3. Conventional log classification & debug method

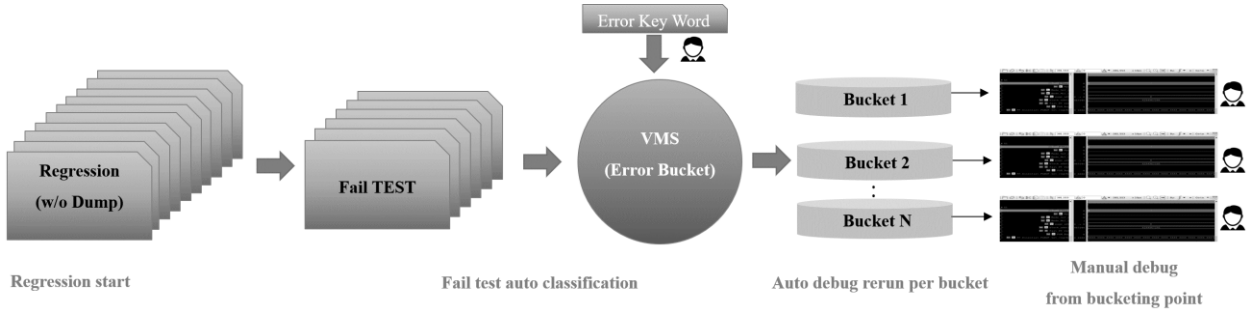

Figure 4. Proposed log classification & debug method

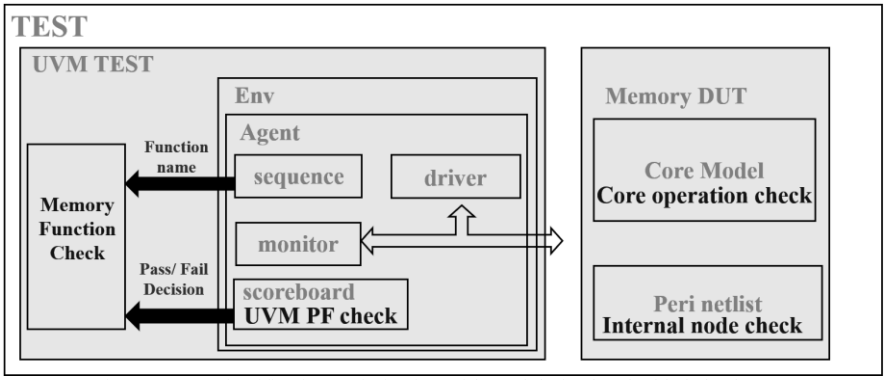

Figure 5. Log classification optimization with modularized embedded checkers

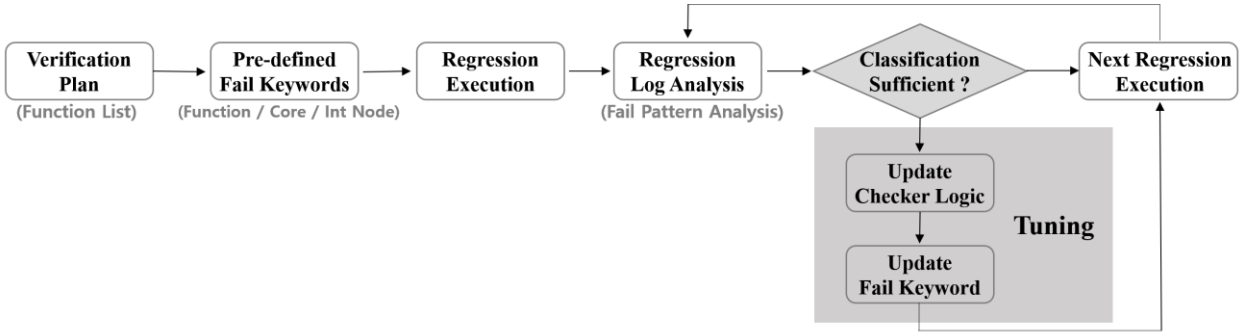

Figure 6. Plan-driven fail keyword definition and tuning flow in the proposed method

C. Enhanced Linkage Between Simulation Logs and Waveforms

Enhancing the correlation between simulation logs and waveforms can improve debugging efficiency. A checker was configured to record the failure keyword, instance location, and simulation time of each failure in the log. When a failure log occurred during regression, the commercial EDA tool's Regression Debug Automation (RDA) feature automatically extracted the instance path, timestamp, and relevant signal list based on predefined fail keywords. This information has been directly linked to the waveform viewer, allowing engineers to start debugging immediately from the failure instance and simulation time. As a result, unnecessary manual searches have been eliminated, and the time required for root-cause analysis has been significantly reduced.

Fig. 7 shows how a commercial EDA tool generates fail buckets and provides RDA reports. Representative benches have been automatically rerun for debug, and through the GUI "Waveform Analysis" command, the waveform viewer is launched automatically with the corresponding FSDB file and predefined signal list pre-loaded. In addition, schematic–waveform cross-probing enables engineers to trace the root cause efficiently from the failure point.

D. Incremental Regression Failure Log Analysis

In a conventional regression environment, as shown in Fig. 8, engineers manually monitor regression progress until a failure occurs, then identify the failure and manually initiate a debug rerun. In this workflow, debug reruns can only begin after the engineer detects the failure, resulting in significant delay before the FSDB waveform file is generated. Furthermore, generating the FSDB for all failed tests places a significant load on LSF cluster resources and consumes significant disk space.

We address these challenges by adopting the incremental debugging feature of a commercial EDA solution, SNPS VMS + Regression Debug Automation (RDA). As shown in Fig. 9, this approach automatically executes parallel debug reruns and performs bucketing (failure clustering) based on error/failure log analysis when a failure is detected within a regression set. Bucketing is performed in real time, so we share the same root cause and tests grouped into a single bucket require only one FSDB to be generated. Additionally, at the user's option, a test can be immediately killed and bucketed as soon as an error occurs. The DF (Debug Facilitator) engine is executed immediately, and it will prepare debug databases such as waveform, simulation state, and UVM transactions. These allow engineers to initiate failure case debugging while the rest of the regression runs. Overall, verification engineer can reduce debugging processing time and ease the burden on machine resources and disk space.

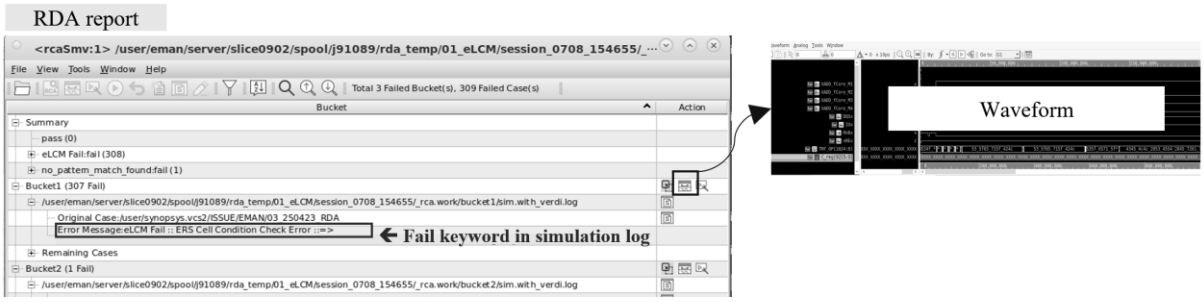

Figure 7. Simulation log-waveform correlation via RDA report

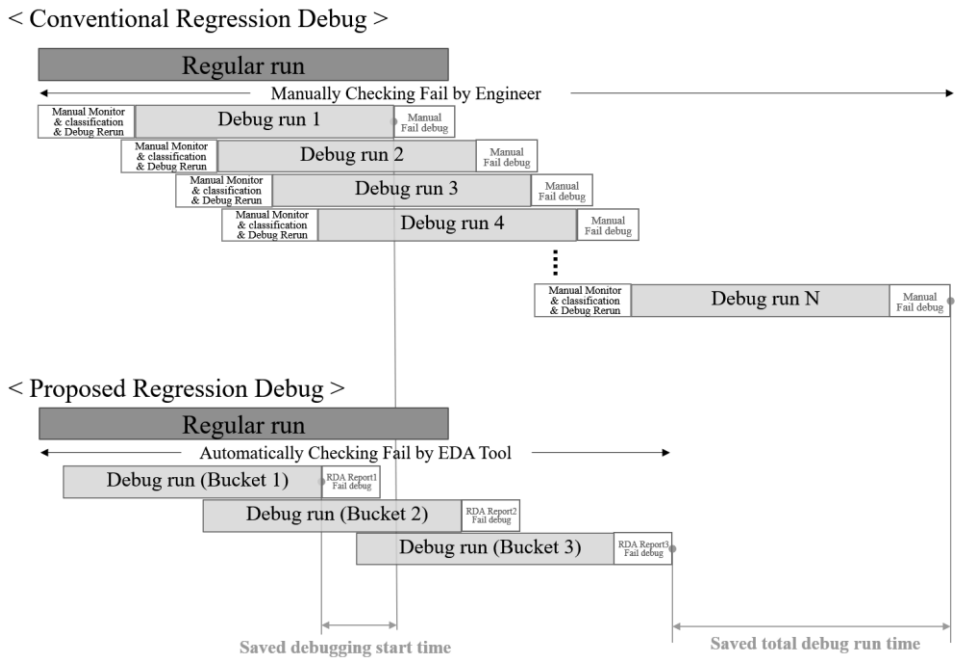

Figure 8. Comparison of conventional and proposed regression debug flows

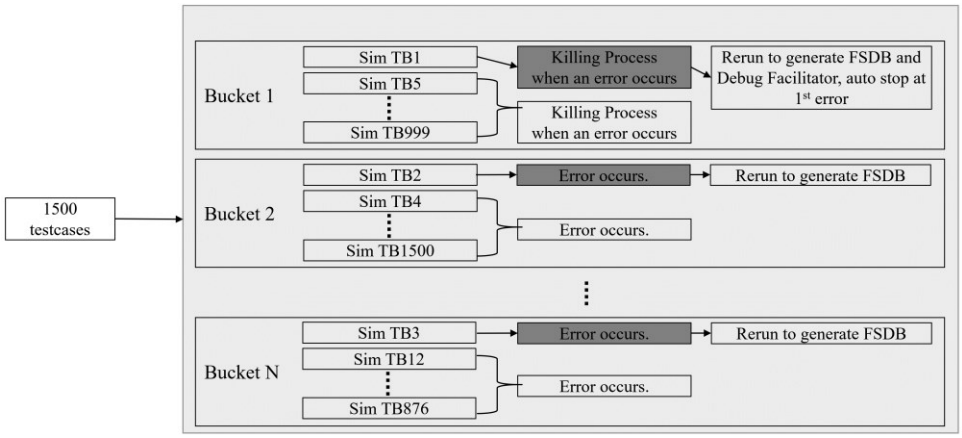

Figure 9. Incremental binning in the proposed flow

E. Customized Environment Utilizing VMS Flexibility

During regression, verification engineers often wish to customize the environment to improve verification efficiency. In our verification environment, each test produces two logs: a Host log and a Test log. The Host log records the pass/fail outcome of each sequence (Seq) as it completes, while the Test log is a conventional simulation log. The final pass/fail result for a test is determined by examining both logs. In this context, we found it beneficial to introduce additional statuses beyond pass/fail—specifically, Stuck and Runtime Fail. We define Stuck as a state in which the test continues to run, but no log updates occur for longer than a specified threshold. Runtime Fail refers to a case where the Host log has already recorded a Fail for some sequence, yet the simulation is still progressing and the Test log continues to be produced. As shown in Fig. 10, although VMS does not natively support these statuses, we enabled them through customization, allowing engineers to construct a regression environment in VMS that better matches their verification needs. For Runtime Fail cases, where further simulation is unnecessary once a Fail has been recorded in the Host log, engineers can kill or re-run the test, reducing the need for engineers to monitor logs in real time.

In addition, as shown in Fig. 11, GUI customization enables engineers to create needed buttons and present useful information. With a single click in the GUI, verification engineers can open the RDA report (bucketized results) and

launch a debugger tool to begin debugging. We also display, within the GUI, the available disk space for each project account so that engineers can reference disk resources when running regressions. These capabilities do not require modifying complex internal tool code; engineers can implement them by writing simple java-script and registering it with VMS. Furthermore, various JavaScript examples—such as disk usage and coverage trend—are included within the VMS. Engineers can leverage these examples for maintenance or extend them to add new functionality.

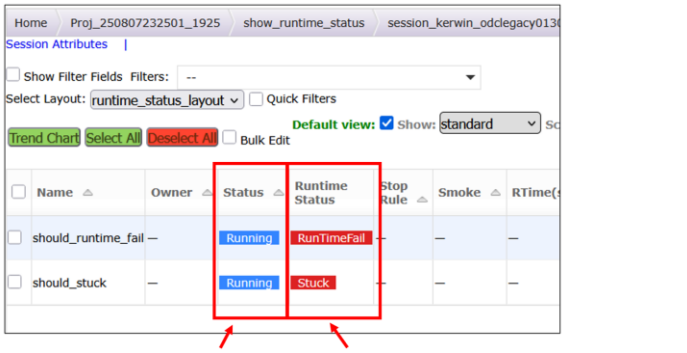

Figure 10. Customized GUI (custom status)

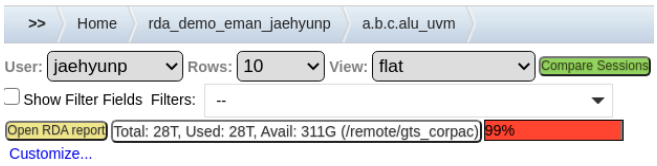

Figure 11. RDA report button and capacity display feature

IV. Experimental Results

To evaluate the effectiveness of the proposed approach, regression experiments containing 1,500 testbenches were performed and compared against the conventional flow. Fig. 12 shows the distribution of simulation runtime. The minimum runtime was 846 seconds and the maximum runtime reached 707,838 seconds (approximately 8.2 days), meaning that the maximum runtime determines the turn-around time for a single regression. To use the verification methodology discussed in this paper leveraging SNPS VMS Tool (VC Execution Manager) and RDA Tool (Verdi), one VC-EMAN-SERVER license is required, along with one VC-EMAN-CLIENT license for each regression set. Additionally, a Verdi license is needed for RDA usage.

In the conventional flow, each fail log had to be manually analyzed, and debug reruns were individually executed. When failures occurred in 100 testbenches, engineers needed to manually inspect all 100 logs, often debugging redundant failures caused by the same root cause. This process increased storage usage and consumed significant manual triage time. As shown in Fig. 13, the first regression typically involved debugging short-runtime functional tests manually; the second regression reran all testbenches to debug additional failures; and the third regression verified and fixed the remaining issues.

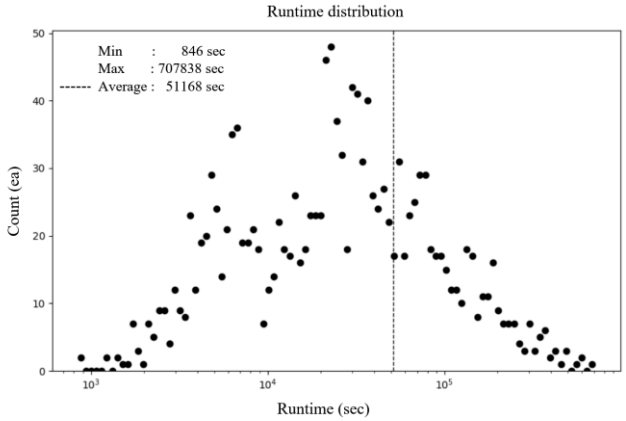

Figure 12. Simulation runtime distribution of 1500 regression testbenches

In contrast, as shown in Fig. 14, the proposed flow defined fail keywords during the initial setup and automatically classified failures by functional category during regression. In this experiment, 136 memory functions were subdivided, and failing testbenches were automatically grouped into eight buckets. A representative testbench from each bucket was automatically rerun for debug, and the necessary wave dump files for root-cause analysis were generated without engineer intervention. Although some buckets contained multiple root causes and a second regression was occasionally required, repetitive manual tasks such as classification and rerun-queue management were eliminated, making the third regression unnecessary. Consequently, the total debug time was reduced by approximately 33%. The enhanced log–waveform correlation also enabled automatic loading of predefined signals at the failure instance, allowing engineers to begin debugging immediately based on fail keywords. This feature has provided a faster and more focused approach to root-cause identification compared to conventional exploration-based debugging.

Table I compares the proposed integrated platform with the conventional environment. The proposed platform unified five previously separate processes—simulation monitoring, log analysis, debug rerun, waveform analysis, and coverage analysis—into a single environment, resulting in a more efficient workflow. As shown in Fig. 15, by inserting memory-function, core operation and UVM PF checkers, the system automatically classified fail logs into 157 buckets.

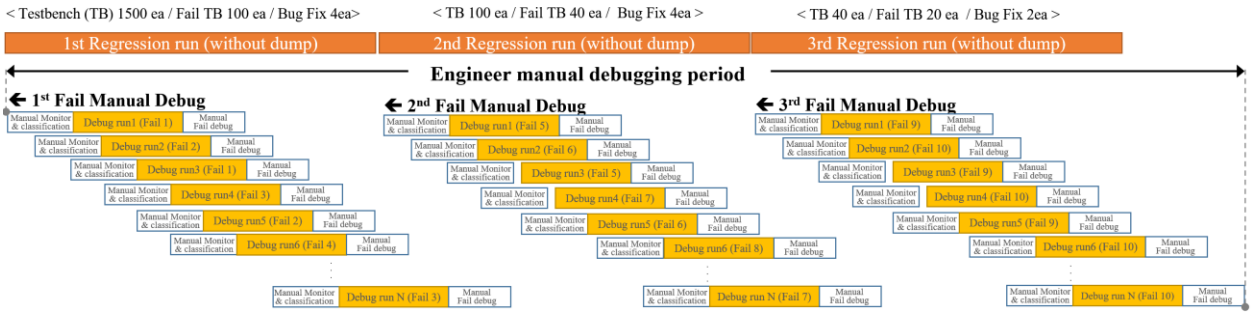

Figure 13. Conventional verification flow regression results

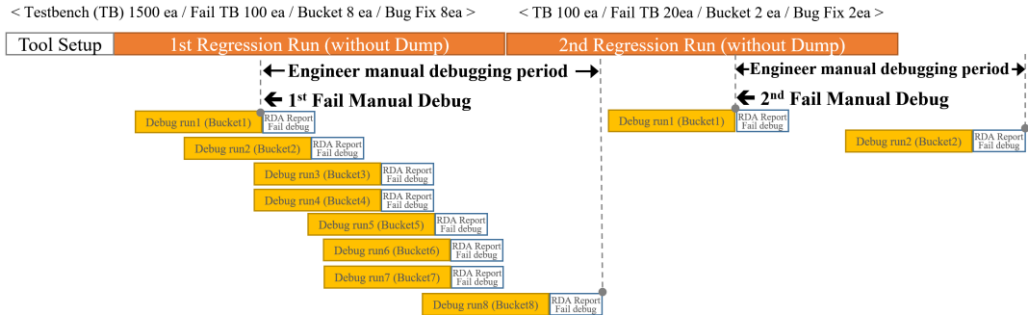

Figure 14. Proposed verification flow regression results

TABLE I
COMPARISON OF VERIFICATION ENVIRONMENTS

Category	Conventional	Proposed
Platforms for debugging (ea)	5	1
Bucket generation method (way)	1 (UVM PF)	3 (UVM PF, Function, Core Operation)
Auto-Gen buckets by UVM PF check (ea)	1	1
Auto-Gen buckets by function check (ea)	0	136
Auto-Gen buckets by core operation check (ea)	0	20
Total Auto-Gen buckets (ea)	1	157

Table II summarizes the debugging TAT and the period of manual debugging effort in the conventional and proposed approaches. As shown in Fig. 16, the proposed platform has reduced manual debugging effort by roughly 50%, decreased regression iterations from three to two, and shortened total debug TAT by 33%. Consequently, the debug productivity of verification engineers has improved by approximately 1.5×, achieving a reproducible and measurable reduction in TAT.

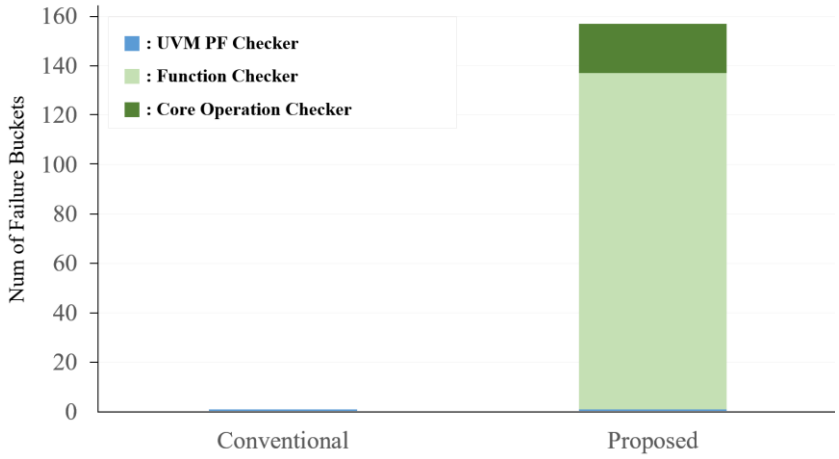

Figure 15. Comparison of Auto-Generated Failure Buckets

TABLE II
IMPROVEMENT IN REGRESSION AND DEBUG TURN-AROUND TIME

Category	Conventional	Proposed	Improvement
Regression count	3	2	↓33%
[a]Total debug TAT (days)	24.6	16.4	↓33%
Engineer manual debugging period (days)	24.6	12.3	↓50%

[a]One regression = 8.2 days (max runtime).

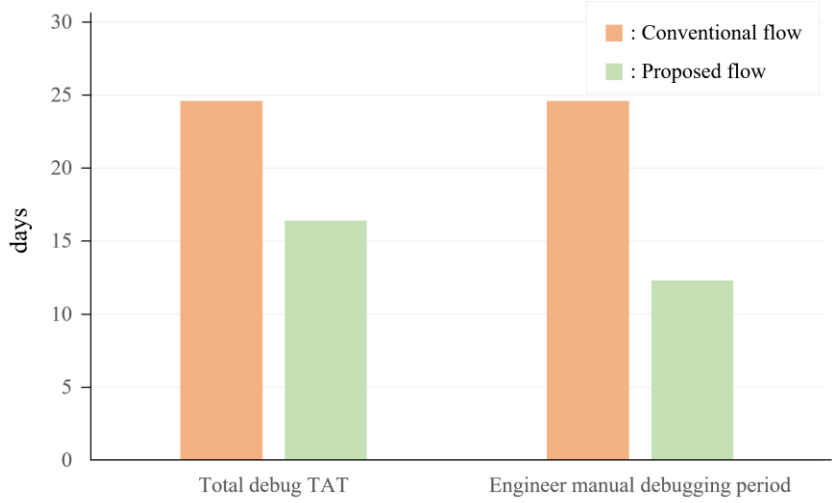

Figure 16. Regression and Debug TAT Comparison

V. Conclusion

An integrated verification platform has been developed to maximize debug efficiency during regression testing. The platform combines log analysis, automatic debug rerun, waveform correlation, and coverage analysis into a unified environment, eliminating unnecessary tool switching and manual operations. By introducing a modularized checker structure—comprising Function Check, Core Operation Check, and UVM PF Check—the system enables automated fail classification at the function level and direct correlation between simulation logs and waveforms. This allows engineers to begin root-cause analysis immediately from the failure instance, greatly reducing debug latency. Experimental results have shown that the proposed platform achieved a 33% reduction in total debug TAT and improved verification engineer productivity by approximately 1.5× compared to the conventional flow. Although the proposed framework was validated using Flash memory regression, its architecture is not limited to a specific memory type. The same approach can be extended to other memory and SoC verification domains that exhibit repetitive debug patterns, deep protocol stacks, and large regression volumes, such as DRAM, controller IPs, and subsystem-level verification. More importantly, this paper illustrates a standardizable automated debug workflow that combines commercial EDA capabilities with domain-specific checkers, rather than relying on ad-hoc scripts or individual expertise. Therefore, the proposed platform enables verification teams to adopt automated debug flows with minimal customization, allowing immediate practical application in real-world verification environments.

References

[1] S. Safarpour, E. Qin, Y. Yang, and B. Keng, "Failure Triage: The Neglected Debugging Problem", DVCon US, 2012.

[2] G. Allan, "Evolution of Triage: Real Time Improvements in Debug Productivity", DVCon US, 2016.

[3] M. Moness, L. Gaber, A.I. Hussein and H.M. Ali, "Automated Design Error Debugging of Digital VLSI Circuits", Journal of Systems Architecture, 2022.