A Novel Fast Regression: An AI/ML Driven Automated Sanity Regression Flow

Joonho Chung[1], Daeseo Cha[2], Woojoo Space Kim[3], Youngsik Kim[4], Seonil Brian Choi[5], Nili Segal[6]

Samsung Electronics Co. Ltd., Seoul, Korea
([1]june26.chung; [2]dscha; [3]space.kim; [4]ys31.kim; [5]seonilb.choi@samsung.com)

Cadence Design Systems, Israel
([6]segal@cadence.com)

***Abstract* – Conventional sanity regression in SoC verification is limited by manual test selection and labor-intensive waveform analysis, delaying feedback after frequent RTL updates. While full regression remains essential for final closure, its long turnaround time (TAT) and high compute cost reduce its practicality during active development. This paper presents Smart Sanity Regression (SSR), an orchestrated AI/ML-driven methodology that integrates isolated verification automation capabilities into a single, end-to-end sanity regression workflow. SSR emphasizes workflow-level orchestration rather than individual AI techniques, unifying change-impact analysis, intelligent testset optimization, failure triage, and waveform-based root-cause analysis under a consistent execution and data handoff framework. Applied to three production IP verification environments, SSR achieves up to 89% regression runtime reduction, 92% compute resource savings, and 56% TAT reduction while maintaining equivalent functional coverage. These results demonstrate that the effectiveness of AI in industrial verification derives primarily from coherent workflow integration aligned with real engineering practices.**

I. Introduction

As System-on-a-Chip (SoC) designs continue to grow in size and complexity, advanced verification techniques are increasingly required to maintain design quality under tight development schedules [1]. Large-scale regression suites are routinely executed after RTL updates, but while full regression remains essential for final verification closure, its long turnaround time (TAT) and high compute cost make it inefficient for frequent design changes [2]. Sanity regression addresses this challenge by executing a compact, high-value subset of tests to provide rapid functional feedback [3]. However, conventional sanity regression flows still rely heavily on manual heuristics and engineer intuition, limiting scalability and consistency. The primary bottlenecks of traditional sanity regression arise from judgment-intensive tasks such as selecting tests that precisely match RTL changes, clustering large volumes of failure logs by root cause, and identifying waveform signals correlated with functional failures. As regression size and design complexity increase, these tasks become increasingly time-consuming and error-prone, directly extending TAT. Static scripts and rule-based heuristics struggle to capture the complex, non-linear relationships between code changes and observed failures. In contrast, AI/ML-based techniques can analyze high-dimensional verification data—including source code changes, simulation logs, and waveform behaviors—to identify correlations that do not scale with manual analysis. Modern verification platforms, including Cadence Verisium Apps, provide AI-driven capabilities for change-aware test selection, automated failure clustering, and waveform-based analysis, effectively addressing individual pain points in sanity regression [4]. However, when used as standalone tools, they still require engineers to manually coordinate execution order, data handoff, rerun policies, and debug workflows, resulting in a fragmented process that limits the overall impact of automation.

This paper presents Smart Sanity Regression (SSR), an orchestrated sanity regression methodology that elevates existing AI/ML-based verification capabilities into a single, coherent workflow. The novelty of SSR lies not in introducing new AI algorithms, but in architecting an orchestration layer that encodes verification intent, execution policies, and cross-stage data continuity. By systematically connecting change-impact analysis, intelligent test selection, failure triage, rerun orchestration, and waveform-based root-cause analysis, SSR transforms sanity regression into a repeatable and engineer-independent methodology. An important aspect of this work is that SSR was developed through close collaboration with the Cadence Verisium engineering team during active tool development. Production-driven requirements and real verification workflows from multiple IP projects were continuously reflected in tool interfaces and automation boundaries, ensuring operational practicality in large-scale industrial environments. Applied to multiple production IP verification projects, SSR significantly reduces regression turnaround time and manual effort while preserving functional coverage on modified modules, delivering fast and consistent verification feedback immediately after RTL changes.

II. Background and Challenges

As System-on-a-Chip (SoC) designs continue to increase in complexity, regression-based verification has become essential for ensuring functional integrity. In production IP environments, each regression cycle may take one day to one week and execute hundreds to thousands of tests, often consuming substantial compute resources and tool licenses [5]. While full regression is indispensable for final verification closure, its long runtime and high compute cost make it inefficient for rapidly assessing the impact of frequent RTL updates [6],[7]. Sanity regression mitigates this limitation by executing a compact, high-value subset of tests to provide early functional feedback. However, in practice, existing sanity regression flows still rely heavily on manual operations, resulting in inefficiency, inconsistency, and high human dependency.

The overall flow and potential for human error in conventional sanity regression is shown in Figure 1. It consists of four major stages: test selection, execution, failure triage and rerun, and waveform debugging. Manual involvement is required in almost every stage, with failure triage and waveform analysis being the most labor-intensive. In our production IP verification environments, manual sanity regression typically consumes 4 to 8 engineer-hours per iteration, depending on regression size and failure patterns. A substantial portion of this effort is spent on repetitive log inspection, rerun management, and waveform navigation rather than on actual root-cause analysis. Such heavy reliance on manual effort not only inflates the overall turnaround time (TAT) but also leads to inconsistent results across engineers.

Three major issues remain in conventional sanity regression:

- **Test selection:** Predefined test lists are commonly used without fully reflecting RTL changes, resulting in redundant simulations and extended TAT. Conversely, critical tests may be excluded, delaying bug discovery and prolonging debug cycles.
- **Failure triage:** Log inspection and clustering are manual steps that depend heavily on engineer experience and maintenance-intensive in-house scripts. In large-scale regressions, engineers may analyze hundreds of log files per iteration, which can delay critical reruns by several hours and introduce significant variability between engineers.
- **Waveform debugging:** Engineers must manually trace error messages and identify root-cause signals through waveform inspection. This process often requires 1 to 3 hours per failure cluster, consuming the majority of the debug cycle and frequently introducing human error.

Several EDA tools provide partial automation features such as change-aware test selection, failure clustering, or waveform mining. However, these tools typically operate independently and lack integration across regression stages. This fragmented structure requires manual coordination between tools, limiting automation benefits and reducing overall verification efficiency. As verification environments scale, such fragmentation becomes a major bottleneck. These limitations motivate the need for a unified and automated sanity regression methodology that reduces human dependency, preserves verification quality, and delivers consistent feedback independent of individual engineer experience. The next section describes the proposed Smart Sanity Regression (SSR) framework, which addresses these challenges through an end-to-end AI/ML-based automation approach.

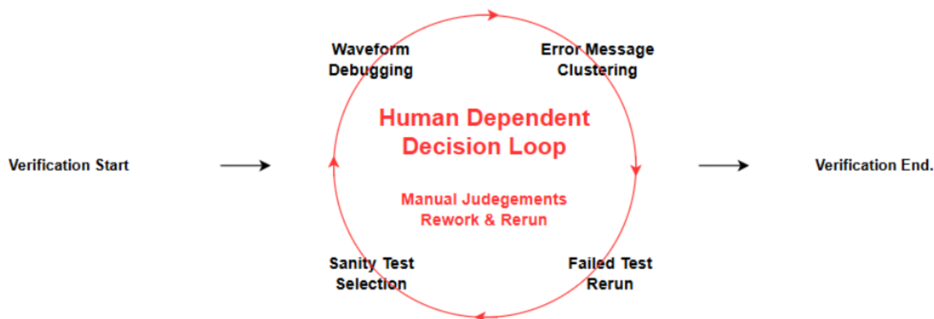

Figure 1. An overview of the conventional sanity regression flow and potential human errors.

III. Proposed Methodology

The Smart Sanity Regression (SSR) framework is proposed as a unified staged pipeline that automates sanity regression after RTL updates (A). SSR integrates multiple verification tools and datasets into a single continuous workflow consisting of targeted test selection (B), automated failure clustering with optimized reruns (C), and waveform-based root-cause analysis (D). These stages are connected through standardized data handoffs, enabling automatic propagation of tool outputs without manual intervention. By unifying these stages into a consistent flow, SSR provides rapid and reliable functional feedback on modified design areas while preserving verification quality and significantly reducing turnaround time (TAT). Figure 2 presents a high-level overview of the methodology.

A. Automated Regression Orchestration Framework

The Smart Sanity Regression (SSR) framework is implemented as an integrated orchestration system that manages the execution environment and data flow of multiple verification tools through a single command. In conventional regression environments, users must manually prepare tool-specific input files, execution parameters, configuration files, and run paths. In contrast, SSR automates this entire setup process by retrieving all required execution information from a reference golden session through internal APIs and automatically generating the necessary configuration files and scripts for each stage. A golden session is a user-specified reference regression session executed prior to the current RTL change and used as a consistent baseline for SSR. As a result, users can launch the complete sanity regression flow with a single command, without manual setup or additional configuration effort. To clarify the end-to-end orchestration process of SSR, Table 1 summarizes the control flow and ex across all stages of the proposed workflow. Once initiated, the SSR pipeline executes a tightly coupled, stage-based workflow with standardized data handoff. In Stage 1, RTL changes are analyzed and a change-aware sanity test set is generated. The resulting artifacts are stored in a standardized directory structure and vsif-compatible format, which are automatically consumed by Stage 2. In Stage 2, regression results and failure logs are clustered within a shared workspace, and representative tests are selected for waveform-enabled reruns. All inputs required for Stage 3—including test lists, waveform mappings, execution parameters, and configuration files—are automatically generated by the orchestration layer upon completion of Stage 2. All stages are centrally managed by a command-level orchestration layer that enforces execution order and data continuity. Each stage triggers the next through explicit completion signals, while data handoff is performed within a hierarchical workspace structure to ensure consistency and reproducibility across runs. Additionally, run-time data is collected into an internal database to support continuous feedback. The following sections describe each stage in detail based on this integrated orchestration architecture.

TABLE 1
ORCHESTRATION SUMMARY OF THE SSR WORKFLOW

Stage	Input	Automated Outcome	Orchestration Role
Stage 1 – Test Selection and Run	RTL diff, coverage, golden session	Change-aware sanity test list	Replaces manual test selection with a consistent, change-driven policy
Stage 2 – Failure Triage and rerun	Sanity regression results, failure logs	Representative failed tests with waveforms	Standardizes triage and rerun decisions to minimize redundant runs and waveform overhead
Stage 3 – Waveform Debug	Failed and golden waveforms	Ranked root-cause signals	Focuses debug effort by prioritizing signals and time windows most correlated with failures

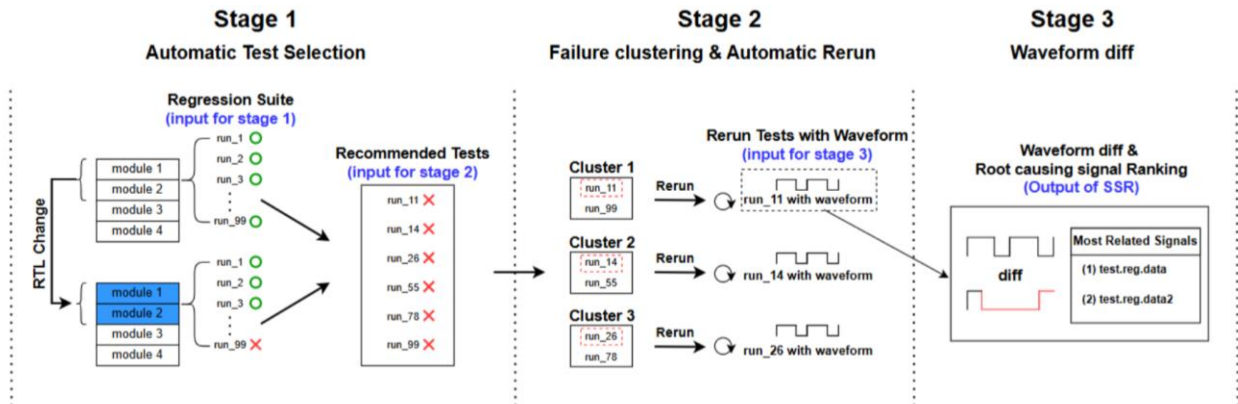

Figure 2. An overview of the proposed flow.

B. Stage 1. Automatic Test Selection

Stage 1 starts with the information of the golden regression session performed before the RTL change. From this baseline, when user runs the command *srun <prj> -ssr=<golden_session_name>*, the orchestrator automatically identifies the required directories and generates the input files needed to execute SSR. First, using the golden regression session information along with the generated files and directories, Verisium AutoFocus generates compact, focused regressions to validate design changes by leveraging test-specific coverage data and test scenarios associated with functional code changes identified by Verisium CodeMiner. This automatic sanity test suite selection provides low-overhead feedback early on and excludes tests unrelated to the RTL changes, ensuring both reliability and quality of sanity test suite despite the short run-time. After completing sanity test suite selection, the orchestrator configures the directory structure required to pass the Stage 1 output to Stage 2. It then initiates the execution of the selected test suite and ends Stage 1. Throughout Stage 1 execution, the orchestrator is leveraged to ensure that execution is maintained within practical limits of run-time and computational resources.

C. Stage 2. Failure Clustering & Automatic Rerun

Stage 2 begins after the execution of the sanity test suite. Verisium Manager collects all failed tests and parses their logs to extract key failure signatures, including error messages, assertion failures, and simulation timeframes. To address the limitations of traditional rule-based heuristics—which struggle with novel failure modes and require constant maintenance—SSR employs Verisium AutoTriage, an ML-driven engine that utilizes a hybrid approach combining unsupervised clustering and supervised classification over structured regression features.

This ML strategy is optimized for regression data, which typically contains a mix of recurring failure patterns and novel errors introduced by recent RTL changes. Unsupervised clustering groups failures based on semantic similarity in log tokens and error paths without requiring predefined labels, enabling organization of both known and unseen failures. When labeled data is available, supervised classification leverages historical regression outcomes to prioritize and rank failure clusters, improving triage consistency across runs. Together, these techniques significantly outperform static regular-expression-based approaches in both accuracy and maintainability.

Once failures are clustered into semantically related groups, SSR selects a single representative test per cluster—typically the shortest-runtime case or the first occurrence—and reruns it with waveform dumping enabled. This policy-driven rerun strategy eliminates redundant executions, reducing simulation volume by approximately 90% (as detailed in Section IV), while ensuring that each distinct failure mode produces an actionable waveform.

Upon completion of reruns, the SSR orchestration layer automatically generates a reference waveform from the golden RTL snapshot and aligns it with the failed test waveforms. Simulation options, random seeds, and dump configurations are synchronized to ensure reproducibility, preparing golden-versus-new waveform data for efficient root-cause analysis in Stage 3.

D. Stage 3. Waveform-based smart comparison

Stage 3 starts by leveraging the representative failed test waveforms generated by the orchestrator in Stage 2. SSR uses Verisium WaveMiner to compare each failed waveform against the golden waveform which dumped before RTL changes. The comparison first detects signal transitions and behavioral deviations between the failed and golden waveform. Signature mining is then applied to identify and rank the signals and timepoints most correlated with the observed failure. These ranked results are presented together with their waveform context to enable intuitive root-cause analysis in Verisium Debug GUI as shown in Figure 3. By highlighting the most relevant signals and their associated waveforms, the system delivers a focused set of potential debug targets. This guided debug process reduces the manual efforts and unnecessary analysis for waveform debugging and minimizes human errors.

At the end of Stage 3 SSR also organizes all generated data into a user-friendly structured directory, where each stage's outputs are stored in dedicated subfolders under a unified top-level directory. In addition, key information from every step is automatically extracted and summarized into a single report file. It allows users to review the entire SSR execution at a glance. This summary includes metrics such as the ratio of recommended test to full regression test suite, overall runtime, failed test details, and correlated failure signal information.

As a result, SSR provides a stable, repeatable, and resource-efficient sanity regression process that maximizes both verification efficiency and quality of the sanity regression.

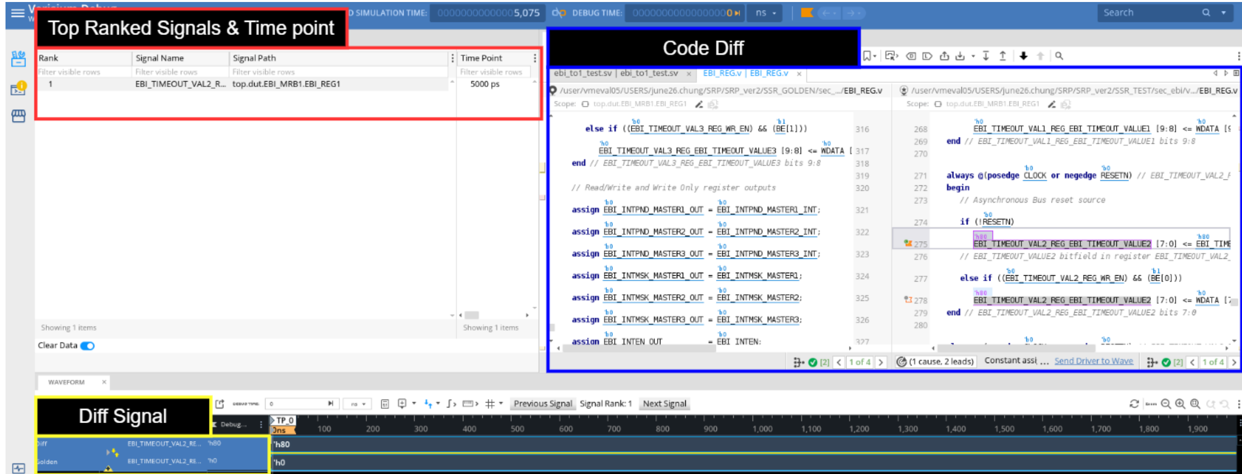

Figure 3. Waveform-based smart comparison Output from Stage 3 of SSR Flow.

IV. Experimental Results

The proposed Smart Sanity Regression (SSR) flow was evaluated in a production IP verification project targeting a complex subsystem within a mobile SoC design. The project environment was characterized by frequent small-scale RTL updates during the development phase, where each change affected only a limited number of modules and interfaces. In such situations, running a full regression after every update can certainly detect functional errors, but it requires substantial computing resources and long TAT, making it impractical for rapid design iterations. To address this inefficiency, SSR was introduced as a lightweight, engineer-invoked sanity regression flow that enables early error detection and functional integrity verification prior to the full regression stage. The methodology was applied across three IPs of varying size and complexity. For each IP, SSR's results were compared with two conventional verification methodologies: a full regression flow without sanity filtering (*A)*, and a manual sanity regression flow managed by verification engineers (*B)*. The comparative results are discussed in detail in the following sections.

A. SSR in comparison with Full Regression

To evaluate the efficiency of the proposed SSR flow, three IPs of different sizes and complexity were analyzed under identical environments. Full regression executed the entire test suite after RTL update, whereas the proposed SSR flow selectively executed the tests correlated with recently modified modules. Figure 4 illustrates the comparative results for test suite size, total runtime, and compute resource usage between the full regression and SSR flows.

A-1. Number of Tests Executed

Across the three IPs, SSR's change-aware test selection reduced the executed test suite from hundreds or even thousands of tests in the full regression to a small subset ranging from 60 to 320 tests, corresponding to only 7–10 % of the total. As shown in Figure 4(a), the reduction ratio was as follows—89 % for IP1, 92 % for IP2, and 93 % for IP3. This consistency demonstrates that the proposed selection mechanism effectively scales with project size, maintaining a stable efficiency gain as the design grows. By compressing the overall regression space nearly ten-fold while still ensuring that all modified design regions are fully exercised, SSR enables a resource-efficient verification process for diverse IP configurations.

A-2. Total Regression Runtime

Across all three IPs, the SSR flow consistently reduced regression runtime by more than 80 %, completing within 2–4 hours on average. IP1 showed the largest relative improvement, cutting the turnaround time from 18 hours to only 2 hours, while IP3 achieved a similar 88 % reduction despite its much larger test suite. This near-linear correlation between test reduction and runtime improvement demonstrates that SSR effectively scales with regression size and provides predictable runtime benefits as project complexity increases. The shortened feedback loop allows verification engineers to confirm design integrity within the same development cycle instead of waiting for full regression completion, greatly accelerating iterative RTL validation. The impact on runtime is presented in Figure 4(b).

A-3. Compute Resource Usage

SSR's selective execution and optimized scheduling significantly lowered the total simulation workload, resulting in more efficient utilization of regression servers. Figure 4(c) shows the comparison of compute-resource usage between the full regression and SSR flows across the three IPs. As summarized in Table 3, SSR consistently reduced compute resource usage by over 90 % across all IPs. For example, IP3 required more than 5,500 core-hours for a full regression but only 450 core-hours when using the SSR flow. This reduction stems not only from fewer test executions but also from eliminating redundant reruns and minimizing simulation overlaps. Such optimization improves regression farm availability and shortens queue times, allowing multiple verification activities to run concurrently. In large-scale environments, this translates into tangible infrastructure savings and enhanced overall project throughput.

A-4. Functional Coverage

Despite the significant reduction in test volume and runtime, the SSR flow achieved 100 % functional coverage on all modified design blocks for every IP evaluated. The coverage results were identical to those of the full regression, confirming that no functional scenarios were lost due to selective test execution. This demonstrates that the SSR methodology maintains verification completeness while substantially improving efficiency, validating its suitability as a fast pre-regression validation flow during active development.

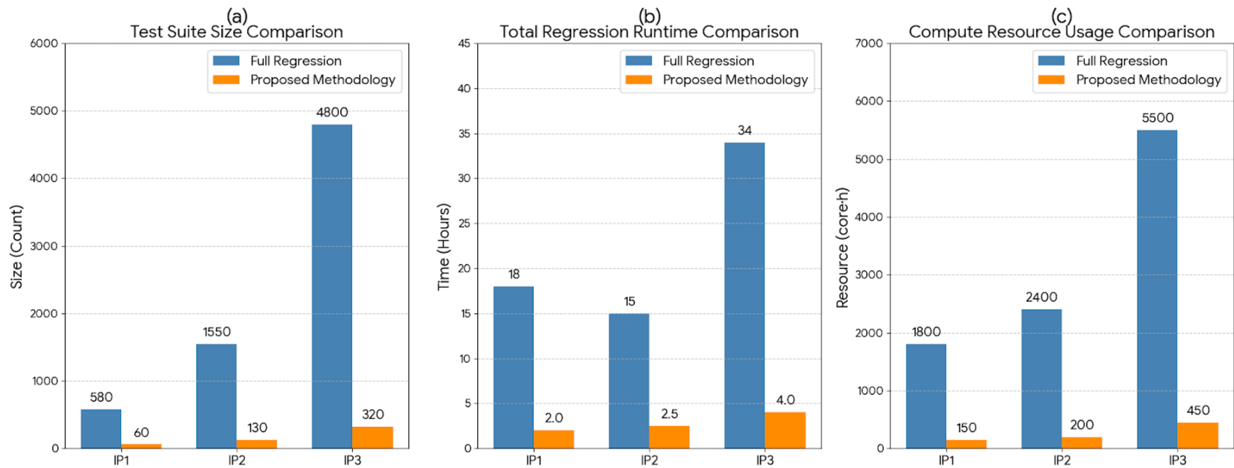

Figure 4. Comparison Between Full Regression and SSR

B. SSR in comparison with Conventional Sanity Regression

In the conventional sanity regression flow, engineers performed every step manually—from test selection, log analysis, rerun scheduling, to waveform debugging. SSR automates these steps in a unified flow, providing change-aware test selections, automated failure clustering with optimized reruns, and waveform debugging. The following sections present comparative results between SSR and conventional sanity regression across three production IPs.

B-1. Stage 1: Test Selection Efficiency

In the conventional flow, predefined sanity suites often contained many tests unrelated to recent RTL modifications, leading to redundant execution and wasted compute resources. SSR automatically analyzed the modification scope and coverage correlation to build a compact, change-aware sanity suite in which all selected tests were directly related to the modified design. Table 2 shows the comparison of predefined and automated test selection across the three IPs.

As summarized in Table 2, predefined suites covered only 10–40% of the modules actually affected by RTL changes, leaving significant verification gaps. In contrast, SSR achieved 100% test relevance across all IPs by automatically correlating tests with modified regions.

Notably, in cases like IP1 and IP3, the SSR-selected test count (320) exceeded the predefined suite (200). This increase does not indicate test bloat, but rather corrects the under-testing of the manual flow, which had failed to cover 95% of the modified logic. By filling these critical verification holes, SSR ensures that every executed test contributes to actual design integrity, unlike static lists that may be smaller but functionally incomplete.

TABLE 2
COMPARISON OF TEST SUITE SELECTION BETWEEN CONVENTIONAL SANITY REGRESSION AND SSR

IP	Predefined Tests	Tests Related to Modification	SSR Tests	Relation to modification
IP1	50	10 (20%)	60	100%
IP2	150	15 (10%)	130	100%
IP3	200	10 (5%)	320	100%

B-2. Stage 2: Failure Clustering and Rerun Optimization

Manual sanity regression required engineers to review each failed test individually, which took several hours and was prone to human errors. SSR automated this process by clustering failed tests with similar log signatures and performing a single representative rerun per cluster. Table 3 summarizes number of failed tests and the resulting clusters across the three IPs. Across all IPs, SSR condensed dozens or even hundreds of failed tests into a small number of semantic clusters—5 for IP1, 8 for IP2, and 15 for IP3—reducing redundant reruns by roughly 90 %. This automation shortened triage time from multiple hours of manual log inspection to only minutes per cluster, allowing engineers to focus on distinct failure categories instead of repetitive error analysis.

TABLE 3
ANALYSIS OF FAILURE CLUSTERING RESULTS GENERATED BY SSR

IP	Failed Tests/Total (%)	Clusters Generated	Reduction in Reruns
IP1	40/60 (67%)	5	88%
IP2	90/130 (69%)	8	91%
IP3	200/320 (63%)	15	92%

B-3. Stage 3: Waveform-Based Smart Comparison and Signal Ranking

In stage 3, SSR performed waveform-based comparison on one representative test per cluster to detect waveform variations against a golden reference. Signals were then ranked by correlation with detected anomalies, providing prioritized debug targets. Table 4 shows the ranking accuracy of the root-cause signal across the three IPs. As shown in Table 4, the signal ranking identified the root-cause signal within the top three recommendations in 50–87 % of cases. This correlation guidance closely matched verification engineer analysis. SSR effectively reduced the average debugging time by directing engineers immediately to relevant waveform intervals and key signals.

TABLE 4
COMPARISON OF SIGNAL RANKING ACCURACY BETWEEN CONVENTIONAL SANITY REGRESSION AND SSR

IP	Clusters Analyzed	Average Rank of True Root Signal	Top-3 Accuracy
IP1	5	3.5	60%
IP2	8	4.7	50%
IP3	15	2.4	87%

B-4. Turnaround time (TAT) Comparison

To measure the overall improvement in TAT, the manual and automated sanity regression flows were compared across all three IPs. Each stage of the manual flow was timed based on engineer execution logs, while SSR's times were automatically collected from the regression scheduler. Figure 5 shows the comparison of total execution time per stage for each IP. Across all IPs, SSR consistently halved total TAT compared with the manual flow, achieving an average reduction of about 55 %. The most significant improvement occurred in Stage 2 (failure triage), where automation reduced analysis and rerun management effort about 66%. This reduction directly contributed to shorter feedback cycles and faster full regression convergence across all evaluated designs.

Across all stages, SSR replaced repetitive manual operations with consistent automation, transforming sanity regression into a predictable, repeatable, and scalable verification methodology. By automating key regression stages including test selection, failure clustering, and waveform analysis. SSR reduced overall sanity-regression TAT by more than 50 %. SSR eliminated redundant reruns, minimized human error, and allowed engineers to focus on root-cause analysis rather than manual data handling. In addition, SSR delivered verification results equivalent to the full regression flow on modified modules while using only about 10% of the tests, runtime and compute resources. By focusing on tests most relevant to each RTL modification, SSR enables engineers to detect errors early and utilize resources efficiently, without waiting for full regression completion or compromising design quality. The combined results from both case studies confirm that SSR significantly enhances verification efficiency while maintaining functional completeness across three IPs.

A limitation of the proposed SSR framework is that its current implementation is tightly integrated with the Cadence verification ecosystem, specifically utilizing Verisium Apps and Verisium Manager for production-level orchestration. While this dependency was leveraged to ensure seamless data access and robust deployment in industrial environments, the core methodology of SSR—workflow orchestration, policy-driven rerun management, and ML-assisted failure triage—is largely vendor-agnostic in concept. These principles can be generalized to other verification environments that provide comparable regression management and debug capabilities. In certain scenarios, such as large-scale RTL refactoring or simultaneous modifications across multiple modules, correlation-based ranking may prioritize secondary effects over the true root cause. Similarly, sparse coverage or incomplete assertions can reduce clustering accuracy, reinforcing the role of SSR as an assistive workflow rather than a fully autonomous debugging solution.

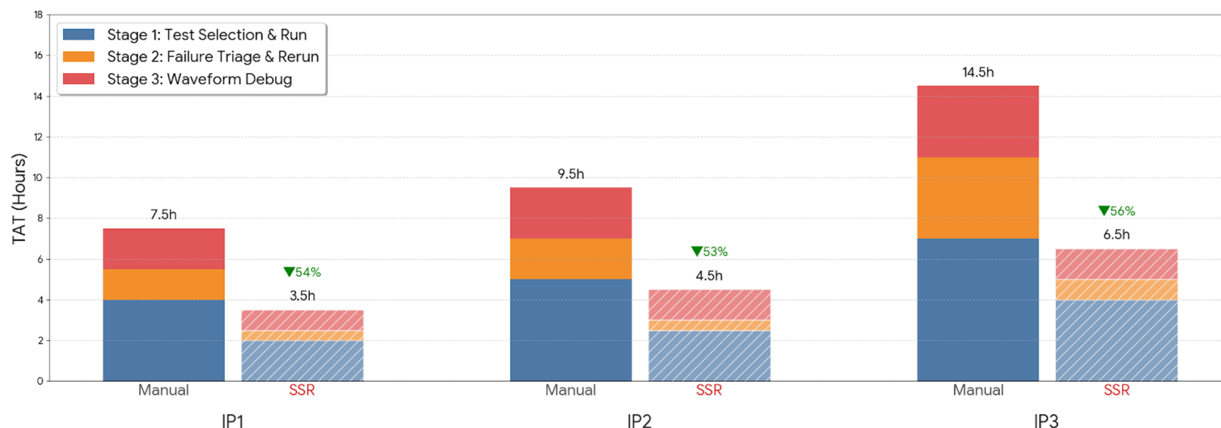

Figure 5. Comparison of TAT Between Conventional Sanity Regression and SSR

V. CONCLUDING REMARKS

Maintaining rapid and consistent verification has become increasingly challenging in modern SoC projects, particularly at the IP level where frequent RTL updates demand immediate validation. This paper presents Smart Sanity Regression (SSR), a unified automation framework that integrates change-aware test selection, machine learning–based failure clustering, and waveform differencing into a single, orchestrated flow. Unlike conventional sanity regression approaches that rely on fragmented tools and extensive manual intervention, SSR enables a one-command execution pipeline with standardized data handoff across all stages.

Beyond raw performance improvements, a key lesson learned from deploying SSR in production environments is that the primary bottleneck in sanity regression lies not in simulation speed, but in human-driven coordination and decision-making. By encoding verification intent, execution policies, and data continuity into an orchestration layer, SSR shifts sanity regression from an experience-dependent activity to a repeatable and predictable workflow. Automating repetitive tasks such as log inspection, rerun management, and waveform setup allows engineers to focus on higher-value activities—including root-cause analysis and testbench enhancement—thereby improving productivity and verification quality while reducing human error. In practice, verification engineers reported that SSR reduced debugging TAT, allowing most sanity iterations to complete within a single working day instead of spanning multiple regression cycles.

Applied to multiple production IPs, the proposed methodology achieved substantial efficiency gains—up to 89% shorter regression runtime, 91% lower compute resource usage, and 54% overall turnaround time (TAT) reduction—while maintaining equivalent functional coverage on all modified modules. These results demonstrate that workflow-level orchestration, rather than isolated tool automation, is critical to achieving scalable and sustainable verification efficiency in industrial environments. Furthermore, the results indicate that SSR is a production-ready solution that reduces engineer dependence and enhances predictability during early-stage IP verification.

While this study focused on IP-level verification, the underlying principles of SSR are broadly applicable to larger verification contexts. Future work will extend the framework to full SoC-level regressions, addressing cross-IP dependencies and shared resource scheduling to further accelerate verification convergence.

For verification practitioners, SSR shows that significant productivity gains can be realized not by introducing entirely new tools, but by systematically orchestrating existing verification automation into a coherent workflow—transforming sanity regression into a fast, reliable, and engineer-independent feedback mechanism.

ACKNOWLEDGMENT

The authors thank Jaeho Jung, Youngnam Yoon, Seungyup Lee for developing the orchestration infrastructure of SSR. The authors thank Jajji Janakiram, Ratan Deep for their continuous support on SSR. The authors are grateful to Lois Lee, Jaejin Lee, Lawrence Loh, Ray Beechinor and Amod Khandekar for their support on Verisium apps.

REFERENCE

[1] J. Bergeron, Writing Testbenches: Functional Verification of HDL Models, 2nd ed. Norwell, MA, USA: Kluwer Academic Publishers, 2003.
[2] "IEEE Standard for System, Software, and Hardware Verification and Validation," *IEEE Std 1012-2016,* pp. 1-260, 2017.
[3] C. Spear and G. Tumbush, System Verilog for Verification: A Guide to Learning the Testbench Language Features, 3rd ed. Springer, 2012.
[4] "Verisium AI-Driven Verification Platform," Cadence Design Systems, [Online].
Available: https://www.cadence.com/en_US/home/tools/system-design-and-verification/ai-driven-verification.html.
[5] A. B. Mehta, "SoC Interconnect Verification," in *ASIC/SoC Functional Design Verification: A Comprehensive Guide to Technologies and Methodologies*, Springer International Publishing, 2018, pp. 273--284.
[6] D.N. Gadde "Improving Simulation Regression Efficiency using a Machine Learning-based Method in *Design Verification" Design and Verification Conference and exhibition*. Europe. Dec 2022.
[7] "Innovative HPC and Verification Technology Speed SoC Development," Cadence Design Systems, [Online]. Available: https://www.cadence.com/en_US/home/resources/technical-briefs/innovative-hpc-and-verification-technology-speed-soc-development-tb.html.