Accelerating Debug with Experience-Driven Insights: A Tool-Based Approach for IP and SoC-Level Verification

Maitreyi Vemulapalli
DCE-NSX
Marvell India Pvt Ltd
Bangalore,India
mvemulapalli@marvell.com

Vinay Datta
DCE-NSX
Marvell India Pvt Ltd
Bangalore,India
vdatta@marvell.com

***Abstract* - As modern SoCs continue to grow in complexity, debugging remains one of the most time-consuming and resource-intensive phases in frontend verification, both at the IP and SoC levels. In large SoC projects,design blocks are a mix of new, reused, and slightly modified IPs. A common challenge arises when DV engineers, particularly those new to a reused block, encounter test failures and must rely on colleagues with prior experience for debug guidance. This dependency often leads to delays, especially when team members are unavailable or distributed across time zones.**

This paper presents a novel, experience-driven debug assistance tool designed to reduce this dependency and accelerate the debug process. The tool captures and stores error signatures observed, along with their most probable root causes, effectively mimicking the knowledge transfer that typically occurs between experienced and new verification engineers. When a test fails, the tool analyzes the failure pattern and provides likely debug pointers based on historical data, enabling engineers to resolve issues faster and more independently.

I. INTRODUCTION

Debugging is a critical yet time-consuming aspect of frontend verification, especially in large SoC projects where design blocks are a mix of new, reused, and slightly modified Ips. This heterogeneous composition introduces a unique challenge: while some verification engineers are deeply familiar with certain blocks, others—especially those working on reused IPs for the first time—face a steep learning curve during debug.

In many cases, engineers rely heavily on colleagues who previously verified the same block to provide debug guidance. This dependency, while natural, introduces inefficiencies:

- **Delays due to unavailability**: Engineers may be busy or in different time zones, leading to long wait times for debug support.
- **Loss of tribal knowledge**: Debug insights are often undocumented and lost when team members transition off the project.
- **Repeated effort**: Similar bugs may be debugged multiple times by different engineers, wasting valuable time.

The result is a bottleneck in the form of prolonged debug time, particularly for engineers unfamiliar with the block's historical issues and quirks. While some teams rely on internal wikis or messaging platforms to share debug knowledge, these methods are unstructured and hard to scale.

To address this challenge, we propose a tool that captures and reuses debug knowledge in the form of error signatures and their most probable root causes.By transforming historical debug data into actionable insights, the tool empowers users to resolve issues independently and efficiently, thus accelerating the overall debug process across both IP and SoC-level verification.

II. TOOL ARCHITECTURE

The proposed debug assistance tool is a Python-based application designed to operate in both GUI and batch (CLI) modes, making it accessible for a wide range of verification workflows. Its primary goal is to reduce debug time and engineer dependency by capturing, storing, and reusing historical debug knowledge in the form of error signatures and their associated causes and fixes.

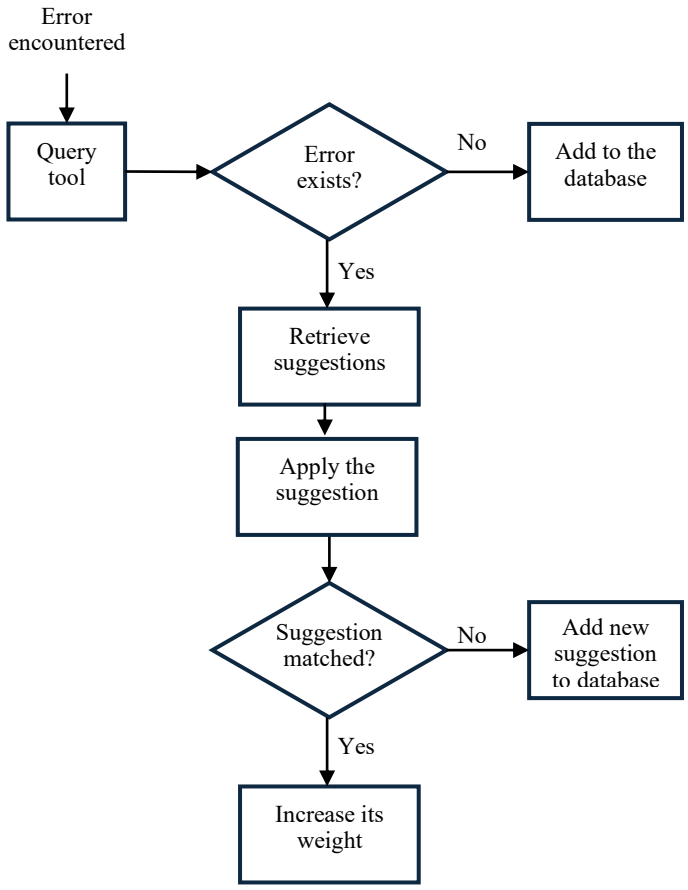
Core Components

- **Error Signature Database**
  A structured repository that stores unique error messages or patterns (signatures) along with one or more associated causes and fixes. Each cause is stored with a weight, representing how frequently it has successfully resolved the issue.
- **Learning Module**
  Engineers can contribute to the tool by adding :
  - New error signatures along with one or more associated causes and fixes.
  - Additional causes for existing error signatures, as new root causes are discovered. Each time a cause successfully resolves an issue, its weight is incremented

  This ensures that the tool evolves with each project and engineer interaction, building a rich, searchable knowledge base over time.

  This collaborative learning loop ensures that the tool remains relevant and effective across projects and teams.
- **Cause Ranking Engine**
  When a known error signature is queried, the tool retrieves all associated causes and fixes, ranked by a weighting system that reflects how often each cause has successfully resolved the issue. This prioritization helps engineers focus on the most likely solutions first.
- **User Interface**
  - GUI Mode: Provides an intuitive interface for browsing, querying, and updating the database.
  - Batch Mode: Allows integration into automated flows or command-line usage for quick lookups and updates.

Workflow Overview



1. Error Encountered: A verification engineer encounters a test failure and extracts the error signature.
2. Query Tool: The engineer inputs the error into the tool (via GUI or CLI).If the error doesn't exist, it gets added to the database.
3. Retrieve Suggestions: If the error signature already exists, the tool returns a ranked list of possible causes and fixes based on historical data.
4. Apply the suggestion: Engineer attempts the suggested fix.
5. Update Tool: If the fix is successful, engineer confirms it, and the tool increments the weight of that cause. If a new cause is discovered, it can be added to the database.

Key Benefits

- Reduces dependency on prior engineers for debug knowledge.

- Accelerates root cause identification by leveraging past experience.
- Continuously improves as more data is added.
- Scales across teams and projects, especially useful in reused IP scenarios.
- Reduces onboarding time for engineers new to a block.
- Minimizes delays caused by engineer unavailability or time zone differences.
- Preserves and scales tribal knowledge across teams and projects.
- Improves consistency and speed in debug practices

III. APPLICATION

The debug assistance tool is designed to seamlessly integrate into a wide range of frontend verification environments, making it highly adaptable to diverse workflows and team structures. It supports both GUI and command-line (batch) modes, allowing engineers to interact with it manually or embed it into automated flows.

Integration into Verification Environments

The tool is verification-environment agnostic and can be easily incorporated into:

- **Regression Scripts:** It can be invoked post-regression to automatically analyze failing tests and suggest likely root causes based on historical debug data.
- **Triage Workflows:** It enhances post-regression triage by helping engineers categorize and prioritize failures using ranked debug suggestions.
- **Interactive Debug Sessions:** Engineers can manually query the tool during active debugging to retrieve probable causes and fixes for observed error signatures.

This flexibility ensures that the tool complements existing infrastructure without requiring major changes. It supports both IP-level and SoC-level verification, and its collaborative, evolving knowledge base makes it especially valuable in projects involving reused or legacy IP blocks.

IV. (PRELIMINARY) RESULTS

A pilot deployment was conducted within a verification team working on a new SoC project with a mix of new and reused IPs. Key observations include:

- Engineers resolved recurring issues independently using the tool's suggested debug pointers.
- The weight-based ranking of causes helped prioritize likely fixes, even when multiple causes were associated with a single error.
- The tool provided appropriate and actionable debug pointers, which helped engineers quickly identify root causes and reduce time-to-fix.
- The tool encouraged a culture of knowledge sharing, where engineers proactively contributed to the growing database.

Though still in its early learning phase, the tool has already demonstrated its potential to reduce debug time and engineer dependency. As the database grows, its accuracy and utility are expected to improve significantly.

V. ANALYSIS

The debug assistance tool presented in this paper addresses a critical gap in frontend verification workflows: the reliance on individual engineer experience for resolving recurring issues. By capturing and reusing debug knowledge in a structured, evolving database, the tool empowers engineers—especially those new to a block—to resolve issues independently and efficiently.

Integration into Existing Verification Environments

One of the key strengths of the tool is its flexibility and ease of integration. Every company typically has its own set of regression management scripts, post-regression triage flows, and custom verification environments. The tool is designed to be agnostic to these setups and can be seamlessly integrated into:

- Regression scripts to automatically analyze failing tests and suggest likely causes.
- Post-regression triage tools to assist in categorizing and prioritizing failures.

- Interactive debug workflows, where engineers can query the tool manually via GUI or CLI.

This makes the tool not just a standalone utility, but a pluggable component that enhances existing verification infrastructure without requiring major changes.

Scalability and Team Collaboration

The tool is inherently scalable—as more engineers contribute to the knowledge base, its effectiveness improves. It also promotes a culture of collaboration, where debug insights are no longer siloed but shared across teams and projects. This is particularly valuable in large organizations with distributed teams and high engineer turnover.

Limitations and Considerations

While the tool shows strong potential, it is currently in its early deployment phase. Some limitations include:

- **Initial cold start**: The tool's usefulness depends on the volume and quality of data it accumulates over time.
- **Error signature variability**: Slight differences in error messages may require normalization or pattern matching to improve hit rates.
- **Manual input dependency**: Engineers must actively contribute to the database for it to grow and remain relevant.

Future Enhancements

To further enhance its capabilities, future versions of the tool could include:

- **Natural language processing (NLP)** to match similar error messages more intelligently.
- **Integration with version control systems** to track changes in design and verification environments.
- **Web-based dashboards** for centralized access and team-wide visibility.
- **Machine learning models** to predict likely causes based on context, not just signature matching.

VI. CONCLUSION

Debugging remains one of the most resource-intensive phases in frontend verification, particularly in environments where IP reuse is common and team members frequently change. This paper introduced a Python-based debug assistance tool designed to reduce engineer dependency and accelerate issue resolution by capturing and reusing historical debug knowledge.

By allowing engineers to store error signatures along with their associated causes and fixes, and by ranking these causes based on real-world success, the tool transforms the debug process into a collaborative, data-driven workflow. Its ability to operate in both GUI and batch modes, and its ease of integration into existing regression and triage flows, make it a practical and scalable solution for modern verification teams.

Although still in its early deployment phase, the tool has already shown promise in reducing debug turnaround time and promoting knowledge sharing across teams. As it continues to learn from real-world usage, its effectiveness will only grow, bringing us closer to a verification environment where past bugs lead to fast fixes.