**JOURNAL OF ASIAN SCIENTIFIC RESEARCH (JOASR)**
2023, Vol. 13, No. 3, May-June, pp. 1-6.
website: https://joasr.com

# Automated Debugging of Hardware Simulation Traces Using Contrastive Learning and Attention Mechanisms

**David Wong,**
Machine Learning Researcher, Canada .

## Abstract

*Debugging complex hardware simulation traces is a labor-intensive and error-prone process due to the intricate nature of digital circuit behavior and trace dependencies. In this paper, we propose a novel approach that leverages contrastive learning and attention mechanisms to automate the identification of fault-prone segments in simulation traces. Our methodology embeds trace segments into a learned vector space using a Siamese architecture, while attention layers prioritize signal relevance dynamically. Experimental results on benchmark circuits demonstrate up to 45% improvement in fault localization accuracy compared to traditional pattern-matching approaches. The proposed system paves the way for efficient trace analysis, reducing manual intervention and accelerating verification cycles.*

**Keywords:** Hardware debugging · Contrastive learning · Attention mechanism · Simulation traces · Fault localization · Machine learning for EDA

**Citation**: Wong, D. (2023). Automated debugging of hardware simulation traces using contrastive learning and attention mechanisms. Journal of Asian Scientific Research (JOASR), 13(3), 1–6.

## 1. Introduction

Modern digital systems are increasingly complex, with billions of transistors and intricate interconnects. Verification and debugging of hardware simulation traces are crucial phases of the design flow, consuming over 70% of the total development time [1]. Simulation traces, representing signal transitions over time, often span millions of time steps and thousands of signals, making manual debugging impractical.

Traditional debugging methods rely on assertion checking, waveform inspection, and log analysis, all of which are manual, time-consuming, and error-prone. Recent advances in machine learning, especially **representation learning**, provide new avenues for automating debugging. This paper explores the application of **contrastive learning** to encode trace segments and **attention mechanisms** to identify signal patterns most relevant to faults. These techniques, when combined, can reveal latent structures in traces and improve fault detection accuracy.

We organize this paper into six sections: background and motivation, literature review, system design, experimental results, analysis and discussion, and conclusion.

## 2. Background and Motivation

Debugging hardware traces involves identifying faulty behaviors by analyzing changes in signal values over simulation time. These traces are often large and noisy, with irrelevant signal transitions masking the actual cause of failure. Prior work has explored symbolic execution [2], logic slicing [3], and anomaly detection [4], but these approaches suffer from scalability and generalization issues.

Contrastive learning is a form of self-supervised learning that learns representations by bringing similar pairs closer and dissimilar pairs apart. Combined with attention mechanisms, which dynamically focus on the most important inputs, these models can learn to isolate relevant features from traces. Our motivation lies in the unique suitability of these methods for handling high-dimensional, time-series simulation data in hardware debugging.

## 3. Literature Review

Recent advancements in machine learning have significantly influenced the field of Electronic Design Automation (EDA), particularly in fault detection and hardware trace analysis. Early works in this domain demonstrated the feasibility of deep learning models to analyze and predict hardware behavior. Wang et al. (2018) investigated machine learning models for performance estimation of EDA flows, showcasing neural networks' ability to learn complex design parameters, though interpretability remained a challenge. Similarly, Lin et al. (2019) explored self-supervised learning for design optimization, paving the way for autonomous debugging systems.

In terms of fault localization, Pradhan and Gupta (1996) proposed stuck-at fault models, a foundational technique in digital testing. However, these traditional models often lack adaptability to the trace data generated during simulation. Later, Patel et al. (2018) introduced an automated diagnosis framework for RTL-level bugs using feature extraction from waveforms, showing improved fault detection but requiring extensive domain-specific feature engineering. Complementarily, Moradi and Sangiovanni-Vincentelli (2020) applied graph neural networks for trace reduction and structural bug localization, addressing scalability and information sparsity.

The incorporation of **attention mechanisms** has been revolutionary in sequence modeling. Vaswani et al. (2017) introduced the transformer model, which leveraged attention to model dependencies across long sequences without recurrence, an architecture that would later inspire attention-based debugging of time-series signals. Lee et al. (2015) also examined temporal models for circuit simulation, which, though lacking attention layers, demonstrated the importance of temporal context in debugging traces.

**Contrastive learning** has emerged as a dominant approach for representation learning, especially when labeled data is scarce. Chen et al. (2020) proposed a simple framework for unsupervised contrastive learning, which was later extended to supervised scenarios by Khosla et al. (2020). These methods showed the ability to learn robust embeddings by bringing similar data pairs closer in the latent space and pushing dissimilar ones apart. This idea has high relevance to hardware debugging, where trace segments can be treated as structured time-series requiring similarity-based comparison.

Trace abstraction, as studied by Hines et al. (2013), is another complementary line of work where behavioral equivalence is used to reduce trace complexity, aiding in better scalability of formal verification and debugging tools. In analog fault diagnosis, Wang and Roychowdhury (2017) successfully demonstrated neural network-based fault detection using simulation data, showing that even non-digital signal behavior can benefit from learned models. Additionally, anomaly detection techniques using generative models, such as those proposed by Ding et al. (2018), provided inspiration for unsupervised trace anomaly detection.

Together, these studies form a strong foundation upon which this paper builds. By combining contrastive representation learning with attention-based signal relevance estimation, the proposed approach aims to improve generalization, fault coverage, and interpretability in automated debugging of simulation traces.

## 4. System Design

The proposed system integrates contrastive learning with attention mechanisms to enable efficient and intelligent analysis of hardware simulation traces. At the heart of the system lies a Siamese neural network that processes pairs of trace segments—each segment composed of multi-signal time-series data—and transforms them into embedding vectors in a latent space. These embeddings capture temporal dependencies and cross-signal relationships using a combination of 1D convolutional layers and gated recurrent units (GRUs), optimized for time-series modeling. Training is guided by a contrastive loss function, where pairs of trace segments labeled as similar (e.g., both fault-prone) are brought closer in the embedding space, while dissimilar ones are pushed apart. This framework helps the model learn subtle variations in faulty vs. clean trace behaviors even without explicit signal-level annotations.

To further enhance the model's interpretability and focus, an attention mechanism is incorporated after the encoding step. The attention layer assigns weights to different signals and time steps, effectively highlighting the most influential parts of the trace responsible for faulty behavior. These attention scores not only improve the model's fault localization capabilities but also provide insights into which signals or time windows are most critical, aiding verification engineers in root-cause analysis. The final classification layer—a shallow feedforward neural network—consumes the attended embeddings and outputs the likelihood of a trace segment being fault-indicative. This modular architecture enables flexibility, supports end-to-end training, and significantly improves the trace debugging workflow in terms of both accuracy and user interpretability.

## 5. Experimental Results

To validate the effectiveness of our proposed model, we conducted experiments using the **ISCAS'89** benchmark circuits, including fault-injected simulations for circuits such as s349, s5378, and s9234. Simulation traces were generated by injecting single and multiple stuck-at faults, and both faulty and fault-free outputs were captured. These traces were segmented into overlapping time windows, each treated as a classification instance. We compared our system with two baselines: a **waveform comparison method**, which identifies differences in signal transitions across time steps, and a **random forest classifier** trained on handcrafted statistical features extracted from trace segments.

Table 2 summarizes performance metrics including accuracy, precision, recall, and a custom-defined **Fault Localization Index (FLI)**, which measures the closeness of the predicted fault signal to the actual injected fault. Our approach outperformed both baselines across all metrics, achieving an accuracy of **88.5%**, compared to **72.4%** for random forests and only **61.2%** for waveform comparisons. Similarly, the proposed model demonstrated a much higher FLI (0.72) than existing methods. Figure 1 plots **accuracy versus fault injection rate**, demonstrating the model's graceful degradation compared to steep declines in baseline performance. As fault injection increased, our model retained robustness due to its attention-based focus on critical signals, while other methods were overwhelmed by irrelevant or noisy transitions.

**Table 1: Performance Comparison Across Methods**

| **Method** | **Accuracy** | **Precision** | **Recall** | **Fault Localization Index** |
|---|---|---|---|---|
| Waveform Comparison | 61.2% | 58.1% | 55.4% | 0.34 |
| Random Forest | 72.4% | 70.2% | 68.8% | 0.49 |
| Proposed (Contrastive + Attn) | **88.5%** | **86.7%** | **84.9%** | **0.72** |

## 6. Analysis and Discussion

The proposed model consistently outperforms baselines across all metrics, especially in trace segments with higher fault injection. The use of **attention** improves interpretability by highlighting key signals, and **contrastive learning** improves robustness to unseen faults.

Limitations include the requirement for labeled traces during initial training and potential overfitting on highly synthetic benchmarks. Future work will focus on domain adaptation and semi-supervised learning to reduce annotation effort.

## 7. Conclusion

We have introduced a hybrid model combining contrastive learning and attention mechanisms for automated debugging of hardware simulation traces. Our approach shows significant improvements in accuracy and interpretability over traditional methods. This study contributes toward more intelligent and scalable verification systems in the hardware design process.

## References

[1] Grohoski, G. T. (2005). Design challenges in the development of the Niagara processor. IEEE Micro, 25(2), 6–16.

[2] Gurushankar, N. (2020). Verification challenge in 3D integrated circuits (IC) design. International Journal of Innovative Research and Creative Technology, 6(1), 1–6. https://doi.org/10.5281/zenodo.14383858

[3] Perry, D., & Kaiser, G. (1990). Adequate testing and object-oriented programming. Journal of Object-Oriented Programming, 2(5), 13–19.

[4] Albinet, A. et al. (2016). A novel dynamic slicing approach for HDL design. In: DATE Conference Proceedings, 12–17.

[5] Balasubramanian, A., & Gurushankar, N. (2019). AI-powered hardware fault detection and self-healing mechanisms. International Journal of Core Engineering & Management, 6(4), 23–30.

[6] Ding, Z. et al. (2018). Anomaly detection for system verification using deep generative models. In: ASP-DAC, 44–49.

[7] Wang, H. et al. (2018). Machine learning for performance modeling of EDA flows. In: DAC, 1–6.

[8] Pradhan, D. K., & Gupta, S. K. (1996). Test generation using stuck-at fault models. IEEE Trans. Computers, 45(9), 1032–1044.

[9] Balasubramanian, A., & Gurushankar, N. (2020). Building secure cybersecurity infrastructure integrating AI and hardware for real-time threat analysis. International Journal of Core Engineering & Management, 6(7), 263–270.

[10] Vaswani, A. et al. (2017). Attention is all you need. In: NIPS, 5998–6008.

[11] Chen, T. et al. (2020). A simple framework for contrastive learning of visual representations. In: ICML, 1597–1607.

[12] Khosla, P. et al. (2020). Supervised contrastive learning. In: NeurIPS, 18661–18673.

[13] Hines, M. et al. (2013). Trace abstraction and its application to bug finding in RTL. In: ICCAD, 432–439.

[14] Wang, Y., & Roychowdhury, J. (2017). Fault detection in analog circuits using deep learning. IEEE Trans. CAD, 36(3), 416–429.

[15] Balasubramanian, A., & Gurushankar, N. (2020). AI-Driven Supply Chain Risk Management: Integrating Hardware and Software for Real-Time Prediction in Critical Industries. International Journal of Innovative Research in Engineering & Multidisciplinary Physical Sciences, 8(3), 1–11.

[16] Lin, Y. et al. (2019). Self-supervised learning in electronic design automation. In:

DATE, 1182–1187.

[17] Moradi, M., & Sangiovanni-Vincentelli, A. (2020). Trace reduction using graph neural networks. In: DAC, 1–6.

[18] Balasubramanian, A., & Gurushankar, N. (2020). Hardware-Enabled AI for Predictive Analytics in the Pharmaceutical Industry. International Journal of Leading Research Publication (IJLRP), 1(4), 1–13.

[19] Patel, P. et al. (2018). Automated trace diagnosis for RTL bugs. In: ICCAD, 34–41.

[20] Lee, B. et al. (2015). Temporal deep learning models for circuit simulation. In: ASP-DAC, 525–530.