# AI-Driven Adaptive Emulation for Accelerated Pre-Silicon Debug: High-Fidelity Silicon Replay

Sampathkumar M Ballary, Aruna Lohiya, Ramesh Patgar, Vandana S, Samruddhi C R, Priyadarshini Panemangalore Pai, Vemuri Krishna Sumanth, Venkata Subba Rao Mane, Subrata Sarkar, Karthik Srinivasan
Samsung Semiconductor India Research, Bangalore, India.
Email: karthik.srvn@samsung.com

***Abstract*- Silicon performance bugs are notoriously difficult to reproduce in pre-silicon emulation. Current platforms, from traditional test benches to modern FPGA-based emulators, fall short of accurately capturing the complex interactions within modern System-On-Chip (SoC) designs. Traffic generators used to mimic traffic behavior of Intellectual Property (IPs) are non-adaptive towards the adhoc system response behavior in meeting bandwidth targets. Furthermore, lack of control over modelling of Memory access latencies across the masters end up most of the silicon issue reproduction at SoC level near impossible in pre-silicon platforms. Also this leads to lengthy manual stimulus tuning, causing delays of weeks or even months. In this paper we propose a novel, fully synthesizable emulation platform to overcome these limitations. By integrating a closed-loop AI-orchestrated system with constraint solvers having the ability to generate stimuli and refine it on the fly over iterations till a given Silicon failure signature is reproduced in a pre-silicon platform. Additionally, this is powered by adaptive learning based Traffic generators and Bus Modelling Engine (modelled memory controller with precise control over system latency behavior). As a result, it converges on the exact architectural state and timing signatures observed in silicon, including read/write latencies for multiple masters. Our benchmarking shows significant improvements: 9x faster corner-case convergence, ~50x faster waveform generation since most aspects of testbench including the Adaptive Traffic generators (modelling the traffic behavior) and Adaptive Bus Modelling Engine (modelling the system latency behavior) are fully synthesizable, ensuring very minimal data exchange between Host and the emulator hardware, also >15% improved memory modeling accuracy compared to state-of-the-art solutions. This work reduces reproduction iterations by >80%, setting a new benchmark for practical and scalable silicon issue debug methodologies.**

## I. Introduction

The evolution of Systems-On-Chip (SoC) has led to increasingly complex functionalities, placing immense pressure on design verification (DV) processes. The challenge lies in identifying silicon performance bugs, which is exacerbated by the limitations of traditional verification methodologies such as static test benches (primarily single or few stimuli based test benches intended for uniform simulation) and Field Programmable Gate Array (FPGA)-emulation platforms. These methods rely on simplistic traffic generators (to generate protocol based traffic such as AXI over time based on the bandwidth defined) and fixed-latency memory models, resulting in inadequate representation of advanced SoC design intricacies. This leads to fidelity gaps, prolonging design cycles, necessitating multiple silicon re-spins, and causing undesirable delays in time-to-market [1]. The advent of Artificial Intelligence (AI) has introduced novel avenues for intelligent design and verification, reportedly holding promising potential for addressing lingering issues in automation, coverage, scalability, and adaptability. Especially, autonomously operating Agentic AI has emerged to facilitate execution of various DV goals with minimal human supervision and process acceleration for complex IPs [2-4]. However, Agentic AI systems face challenges when dealing with incomplete specifications scattered across multiple documents and formats. LLMs may struggle to understand hardware concurrency, timing, and low-level protocol intricacies as deeply as a seasoned verification engineer. Furthermore, lack of adequate context-awareness, scope of integration with proprietary Electronic Design Automation (EDA) flows, project-scale orchestration, and availability of signal-level trace analysis data for root cause identification limit the performance of AI systems [2]. To bridge these gaps, we present a fully-synthesizable Model Context Protocol (MCP) compliant hybrid SoC verification system using multimodal Agentic AI, leveraging adaptive traffic generation and multi-master memory control for close real-time simulation. MCP compliance ensures seamless interoperability and extensibility in synthetic generation, facilitating integration of specialized models for constraint solving, scenario generation, dynamic trigger synthesis, and adaptive refinement.

## II. Problem Statements

Following are the open problems in System Verification aspects targeted from the proposed approach:

a) Shift left Debug platform readiness designed to emulate real-world system behavior and its impact across IPs interacting over interconnects with bare metal (without much dependency of Firmware and Central processing unit (CPU) support to control the hardware) emulation setup during post-silicon validation failures where deep system-wide visibility is required.
b) Low Cost Software infrastructure [Data path intelligent Constraints Solver] to generate required valid and appropriate vectors for enabling multi Subsystem parallel workloads that can be used from Pre-Silicon to Post Silicon Validation platforms.
c) Enable Verification Engineers/Software Development Engineers/Architects who are specialized in a particular domain to verify their Design of interest without worrying about rest of the Design but still make sure their Design is stressed enough from the interaction of region of non-interest too.

d) Lack of accurate Control over Bus Level System parameters including MO (Maximum Outstanding), Transaction acceptance rate, Response Latency (to an IP Client level granularity connected over multi-level hierarchical interconnects) for responses to the DRAM access request initiated by SoC Masters via Network On Chip (NOC) interface through Memory Controller.

e) Most of the pre-silicon tests focus on IP-level and Sub-System Level correctness. However, most silicon failures often arise from unexpected interactions between several components that are too complex and accurate bare metal replay of these interactions is important and difficult since it has several components to behave in the same way as how they behaved on silicon.

## III. Related work

SoC verification has evolved from directed tests to constrained-random traffic using Universal Verification Methodology (UVM). This progress has been accompanied by an increasing infusion of Artificial Intelligence (AI) techniques. Researchers have reported the use of Machine Learning (ML) to improve coverage-directed stimulus selection, learned-branching heuristics, and phase-driven scenario generation. Furthermore, Large Language Model (LLM)-driven frameworks, such as LLM4DV [4], ThreatLens [5], SV-LLM [3] and AssertLLM [6], have been developed to automate various verification tasks. This marks a shift from manual test creation to intelligent, collaborative AI-driven verification processes, which could be expanded to generation of test vectors and adaptive inputs for hardware IPs to converge with signatures even in non-deterministic systems.

In order to validate the SOC Backbone capabilities and to also identify optimized NOC architectures, Traffic Generators play a key role in SOC Architecture Validation and Performance Validation space helping in stressing the system (Bus + Memory backbone) at a pre-silicon stage by injecting traffic into the system. Existing literature disclose protocol complaint [9] configurable traffic generation [8] (parameterized, multiple pattern generation) rather than being self-configurable and adaptive (i.e. runtime autonomous adjustment based on overserved conditions). Furthermore, the focus mostly is on performance verification [10] and [11] rather than debugging anomalous behaviour in SoC which needs precise control over latencies of responses across Masters independently.

## IV. Proposed approach

Proposed framework shown in Fig. 1 targets mentioned problem statements in section[II] that incorporates emulated HW Design platform that takes entire SoC with region of interest/debug having real design and the rest of the SoC IPs stubbed with Adaptive traffic generators. This enabled the ability to replicate the traffic signature from silicon. Furthermore, the infrastructure to generate this design platform is automated.

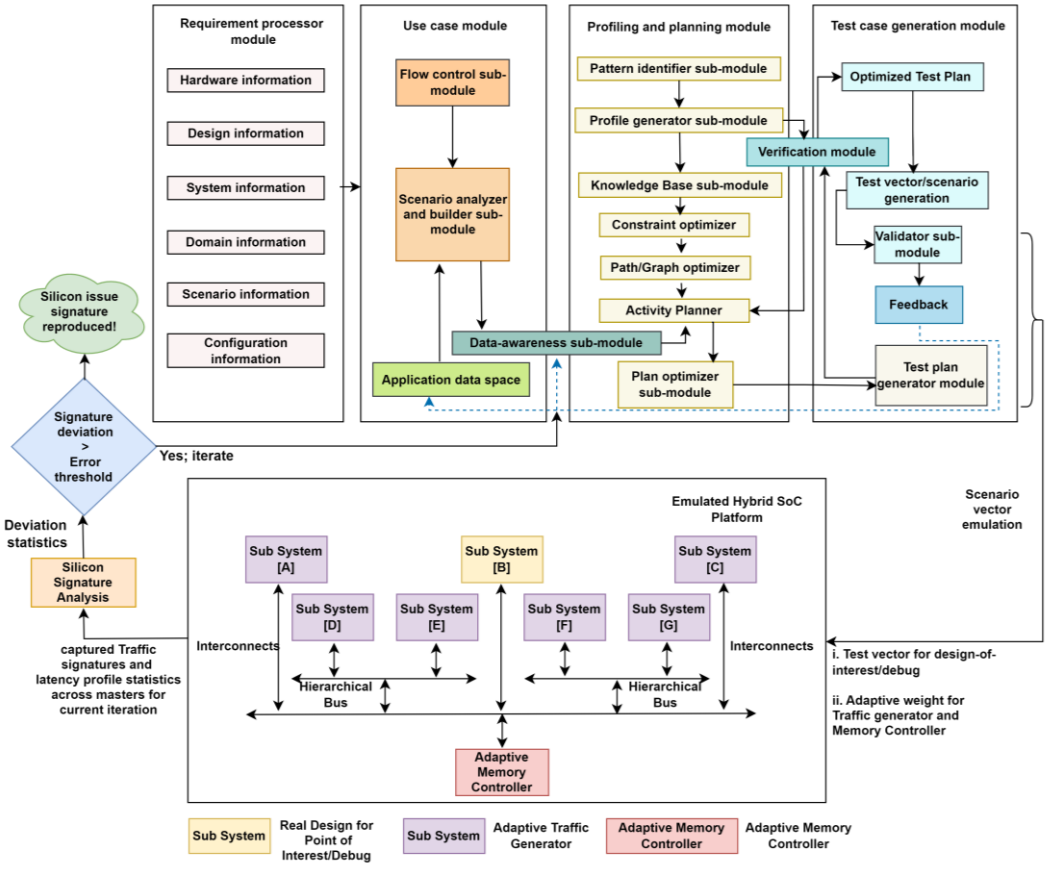


Figure 1. Block diagram of our proposed adaptive silicon replay platform.

The Memory Controller + DRAM system is replaced with synthesizable Adaptive Bus Modelling Engine, which has the ability to replicate Silicon latency profiles with high precision (for both Read and Write Average/Peak latencies independently across 128+ SoC Masters interacting via hierarchical NOCs). Both Traffic Generator and Memory Controller are learning based IPs, whose adaptive weights are powered by Agentic AI involving iterative dynamic constraints solving, which terminates upon convergence of user defined minimal error threshold for Silicon signature reproduction.

At the heart of this framework Fig. 1, there are several interconnected AI agents, each performing a distinct role for execution of various tasks including collection of system information, use-case details, profiling, planning and execution of test case or scenario generation. Constraints depend on various hardware, design, domain and configuration details. To generate constraints with maximum coverage it is important to identify associated information for increasing path awareness during selection of test cases and their execution. This is performed by the profiling and planning module which integrates all the details using an efficient Knowledge Base and optimizers. Initially, the LLM-infusion is verified by domain experts and progressively automated.

*A. Adaptive Traffic Generator*

Traffic Generators are custom Verilog modules that mimic the behavior of specific protocol (like AXI) complaint master to automatically produce streams of read and write transactions on a bus/interconnect. It helps in stressing the system at a pre-silicon stage by injecting traffic into the system. They are also used as part of post-silicon debug to recreate various issues pertaining to performance and backbone architectures.

Major limitations that exist in current Traffic Generators include:

a) Lack of Self Configurable capability to feed in dynamic nature of user requirements over time (t) with user requirements changing across the granular time period (t/N) and inability to solve this problem without any software/external intervention, where N defines the number of intervals into which time (t) can be divided where new user requirements can add.

b) Current Traffic Generators cannot guarantee accurate reproduction of expected IP level behavior due to several variables in the SOC Backbone infra such as traffic from other masters, which are ad-hoc in, hence there is a need to imbibe the ability to dynamic adaptability based on system behavior to mimic expected traffic profile accurately.

To solve above mentioned problems, we have replaced conventional Traffic Generators with Adaptive Traffic Generator which is shown in Fig. 2 is fully synthesizable module that operates in two modes:

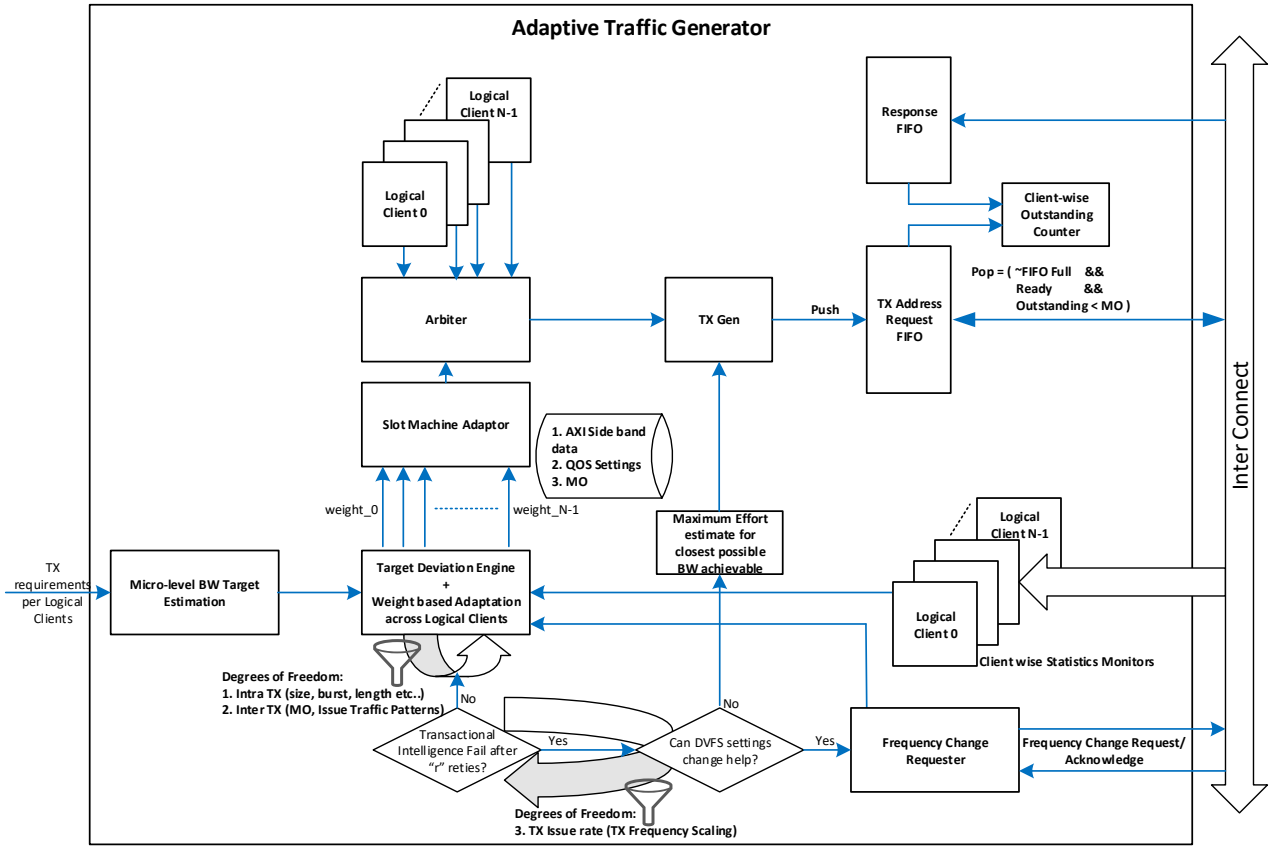


Figure 2. Block Diagram of Adaptive Traffic Generator.

a) Simulation Dump Replay Mode: In this mode, it can reproduce identical results based on provided simulation dump from subsystem based on the silicon scenario.
b) External Target Mode: In this mode, the info on target bandwidth, frequency and latency across a granular time scale is fed into memory and TG will auto fetch at programmed regular interval and keep executing its target requirements and make sure it adapts to the following system factors to meet targets irrespectively.
    i. Transaction Acceptance pattern which emerges as a function of several key parameters - number of Masters interacting in intermediate next system interconnect, their respective Maximum Outstanding Capacities and relative priorities defined among these masters.
    ii. Response rate which is influenced by the key factors - Memory Controller scheduling its memory access pattern and DRAM intrinsic limitations like Self Refresh, R2W/R2R/W2W timing requirements.
    iii. Operating Frequency.

*B. Adaptive Memory Controller/Adaptive Bus Modelling Engine*

Technical problems faced with current Memory Controller + DRAM from silicon issue reproduction point of view can be categorized into 2 main aspects:

i. Lack of Latency Control for Responses to the DRAM access requests initiated by SoC Masters via NOC interface through Memory Controller.
    - Memory Controller, based on its DRAM access pattern and DRAM, based on its intrinsic limitations create random Response Latencies for each of the access request RD/WR, ending up in ZERO control over the Latency factor for all the responses, making it difficult to replicate similar behavior as seen in the silicon dumps.
    - Latency is a major factor that governs the Key Performance Indicators (KPIs) of any SoC Architecture and Performance Validation aspect, which if not controllable, limits many Architectural and Performance related holes present in any SoC.

ii. Accurate and Dynamic control over Latencies across multiple Masters independently for RD and WR access with additional control over addition of Peak value of Latencies per Master interacting via NOC.
    - Any Modern day SoC have lots of Masters interacting via Memory Controller. The KPIs of any SoC depend on these Masters, which are categorized into Real Time (RT) Masters (where the responses to the requests from these Masters are critical w.r.t response latency) and Non-Real Time (NRT) Masters according to the priority assigned and thus Latency becomes a deciding factor for validation. Hence, if this Latency factor is not controllable per Master level independently for WR or RD access, it leads to real bottle neck for Validation.
    - Controllability of Peak Latency addition, over and above Average value control for every Master for both WR and RD accesses independently is a must to validate the SoC Architecture and Performance aspects. Such controllability is needed to account for various conditions, including Multiple Masters interacting simultaneously with very high BW requirements, DRAM requirements of IDLE period like Self-Refresh scheduled by the Memory Controller etc., that end up in surging the Latency value at a peak for particular period and are often the crucial reason of failures in Silicon.

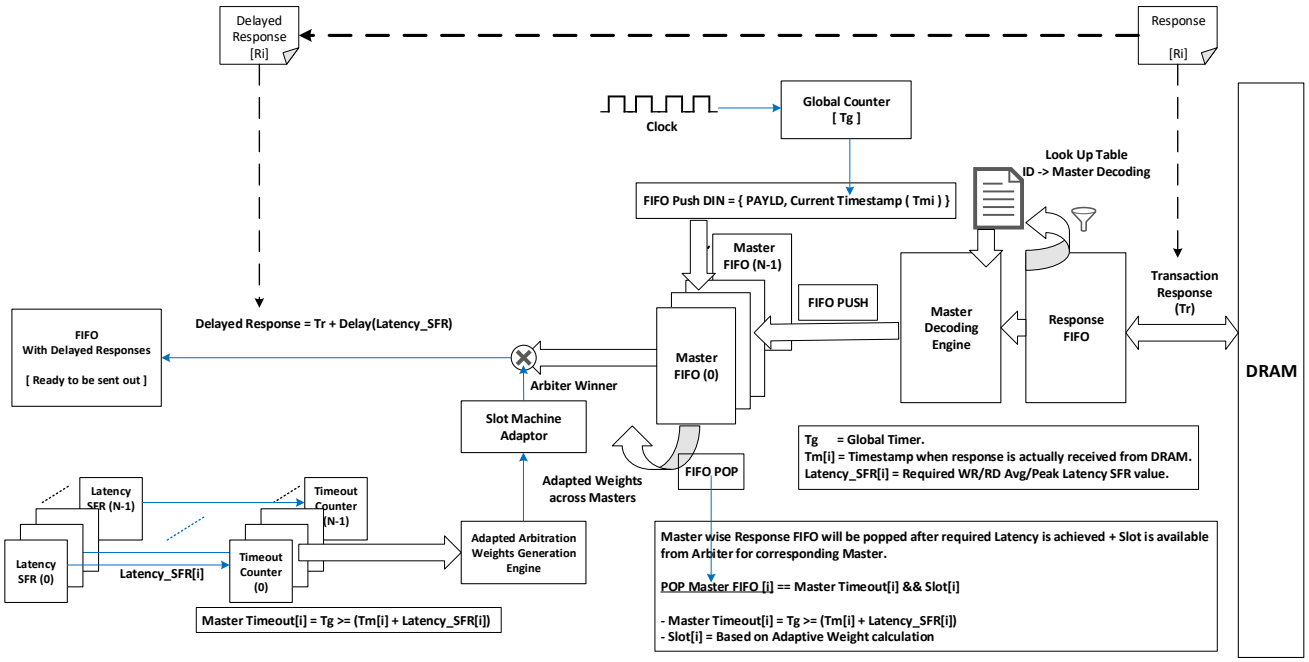


Figure 3. Block Diagram of Adaptive Bus Modelling Engine

Proposed framework shown in Fig. 3 is configurable fully synthesizable Memory Controller with latency control per master (up to 128 unique masters across SoC) independently for WR and RD request with auto tuning ability to adjust the latency error margin across masters, ‘what is expected’ to ‘what is seen’ over a dynamic system behavior with zero software intervention or external tuning agent that can adapt itself in terms of various degrees of freedom like tuning its arbiter scheduling mechanism to meet the latency requirements. This allows the proposed block to be able to achieve accurate modelling of Memory Controller + DRAM with master level accurate w.r.t latency both WR and RD access including peak latency addition over and above average values to model some of the DRAM intrinsic behavior in a highly non-deterministic system.

*C. AI based Constraints Solver and Vector Generator*

AI has been infused in the SOC verification platform using a MCP compliant Agentic AI architecture as shown in Fig. 4. This framework lets different AI models, tools and applications to share context safely and efficiently. Since the ownership of IPs and overall SOC lie with various individuals, we also used this framework to ensure structured and controlled interactions across IPs and components within the Agentic AI platform. Agentic AI can perceive, reason, plan, act and learn autonomously or almost autonomously using internal logic and external tools (like APIs, knowledge bases, or other agents). The information flow across various agents is described below:

i. Orchestrator Agent takes in the constraint file and other details from the system and gives the generated code as an output by coordinating various tasks to be completed by other agents including the XML Agent, Constraints Agent, Graph Agent, Test Agent and Generator Agent.
ii. XML Agent parses the XML input file, extracts configuration details and constructs the structured node tree that is sent to the Constraints Agent.
iii. Constraints Agent simplifies and reformats complex constraints using domain-specific rules, converting them into formats compatible with Satisfiability Modulo Theories (SMT) solvers [7].
iv. The Graph Agent constructs and analyzes subgraphs to detect cyclic dependencies and optimize interconnections among registers.
v. The Test Agent automatically generates unit tests that validate the satisfiability of these constraints via SMT solvers such as Boolector [7]. Error detection is automated to facilitate clean code generation.
vi. The Generator Agent compiles validated constraints into a unified Python-based VSC class or executable scripts, thereby streamlining integration with existing verification environments. It generates test vectors with randomized seeds and provides the same to the Orchestrator Agent for return to the end user.

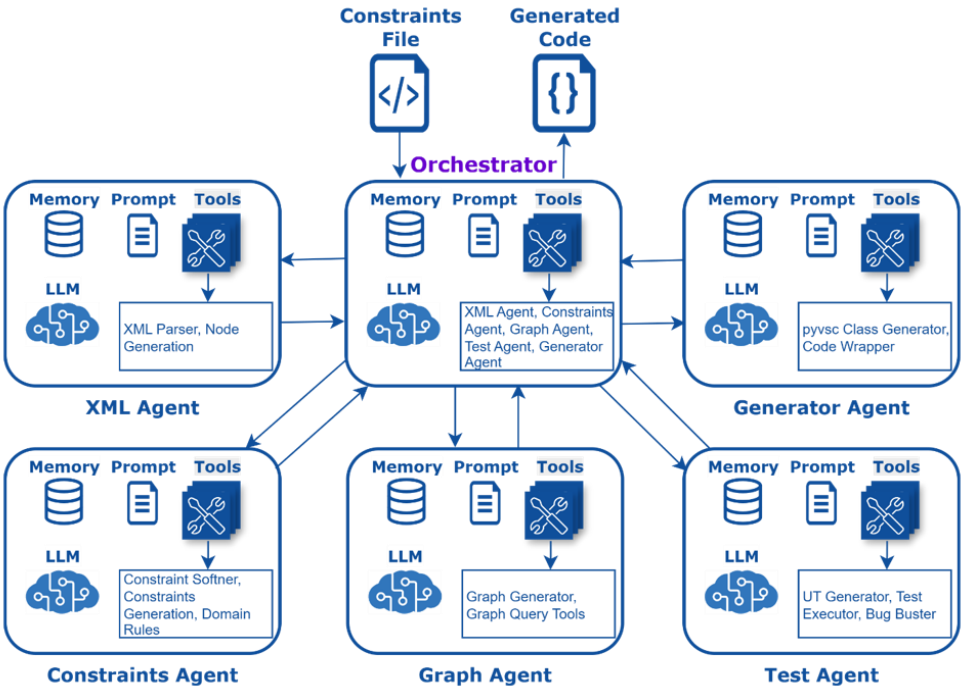


Figure 4. The information flow across various agents for the core constraint solver in the Agentic AI framework

Each of these agents have a set of LLMs pooled from heterogeneous open source foundation models [12], [13], [14] depending upon tasks, memory share and tools associated with the task allocated be it orchestration, parsing, constraint conversion, graph and code generation. They are interconnected via MCP servers which act like a bridge exposing required resources and functionalities diligently to complete the task assigned by the Orchestrator Agent such as querying databases or calling API for collecting system information, use case details, path intelligence, plan generation and feedback.

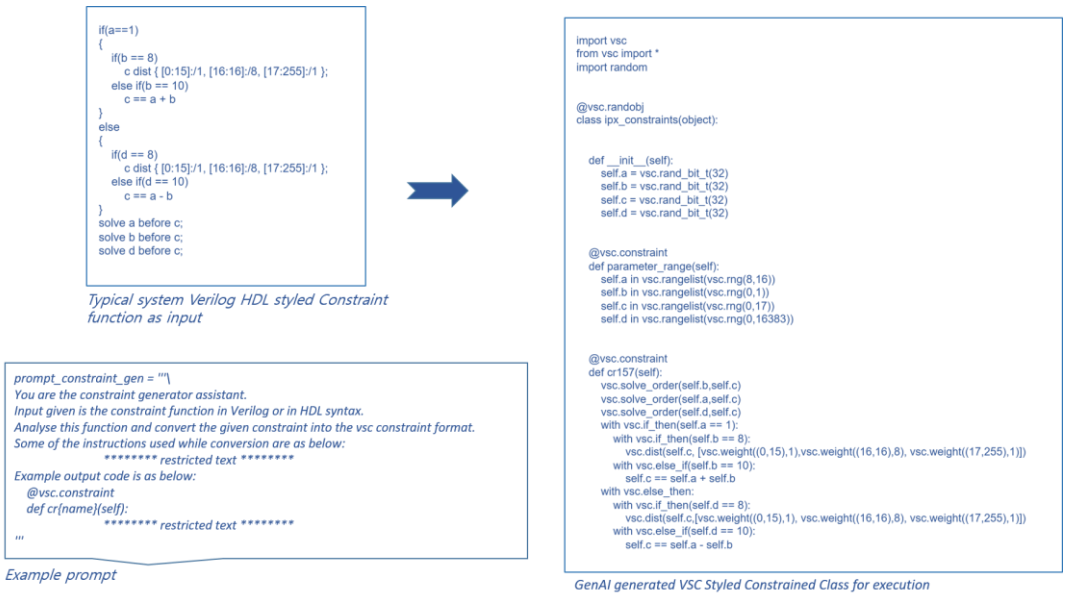


Figure 5. Prompt-based conversion of typical constraint function input to VSC styled constrained class for execution

The MCP server manages input and output schemas, processes prompt templates, and handles requests from clients, returning structured information (one of the example prompt and generated constraints class is captured in Fig. 5). It ensures communication management, often using techniques like chain-of-thought reasoning, such that the correct information is retrieved for the AI. Wherever modules include data diversity, until highly accurate AI performance is achieved, a supplementary approach using Human-In-The-Loop (HITL) is used to ensure prompt validity for maximum coverage and minimal error. Overall proposed architecture stitches use-case based modules with the help of the MCP server allowing all necessary and available information and workflows, powered by generation that is closer to real-world information.

## V. Results

### *A. Reproduced real Silicon Issues using emulated Silicon Replay Platform*

Below are real system failures seen on silicon that were reproduced using silicon replay platform implemented on Palladium emulation setup:

i. Camera Buffer Overflow issue: This is a case where multi Sub-Systems including Camera, Display and other heterogeneous Deep Learning Accelerators being active and due to sudden high bus bandwidth utilization from the Deep Learning Sub System for a particular use case network being executed, hence throttling the Camera Real time DMA path and the increased response latencies for Camera caused the buffer overflow. To reproduce these kind of issues, with limited knowledge on networks being run and the trigger point of same along with the Dynamic Frequency Scaling switches happening regularly, it is important to have Traffic Generator being adaptive to mimic meeting bandwidth targets as well as need for infrastructure to model the latencies seen for each of the Masters not just at Sub System level, but also at the root IP clients level too.

ii. Multi-frame processing Camera IP functional issue: We reproduced a silicon issue on our proposed platform, which was primarily due to a cache pre-fetch design bug in the temporal denoising engine, that resulted in hang state during frame processing. The Silicon issue seen was once in 6 days' case where Camera is kept on continuously and also issue occurs when last of the cache fetch response latency is too high ending up in, one in a million kind of a failure scenario which was easily reproduced in less than 1 day without any prior debug root cause info.

### *B. Analytical results of matching Silicon Signature Bandwidth targets from Adaptive Traffic Generator*

Below are examples showing how different clients in Adaptive Traffic Generator are scheduled by the arbiter using adaptive weights generated from probabilistic history and learns itself to adapt in terms of future weights calculation to meet the bandwidth targets from the silicon signature.

Consider a scenario like mentioned in Table 1:

a) Client0 has more bandwidth to be achieved compared to Client1 and 2, but same bandwidth target as Client3. However, with a difference that Client3 has higher latency for responses compared to Client0.
b) Client1 has similar bandwidth target w.r.t Client2 but having higher latency for responses comparatively.
c) In terms of bandwidth, Client0 == Client3 >> Client1 == Client2.
d) In terms of latency for the responses, Client1 == Client3 >> Client2 ~= Client0.

TABLE 1
SYSTEM TEST SCENARIO BEING EMULATED TO ANALYZE THE ADAPTATION OF TRAFFIC GENERATOR

| Client Number | Number of transactions targeted (in packets) | Number of Clock cycles available to reach target (in clock cycles) | Latency seen for the responses for the sent transactions (in clock cycles) |
|---|---|---|---|
| 0 | 5000 | 40000 | random range(7, 237) |
| 1 | 2500 | 40000 | random range(14, 474) |
| 2 | 2500 | 40000 | random range(9, 239) |
| 3 | 5000 | 40000 | random range(14, 474) |

From a learning point of view over the time, based on the bandwidth and latency profiles, following are some expectations that can be made using Table 1 as reference and Fig. 6 shows how Transaction generator adapted arbitration priority weights across clients based on the bandwidth targets as well as adhoc system responses:

a) Client3 should have higher weighted arbitration slots at the beginning compared to other clients, followed by Client0, Client1 and Client2.

b) Over time, based on the target requirements being met for a particular client, these weights should be adapted to give priority to those lagging behind in meeting their bandwidth targets comparatively.

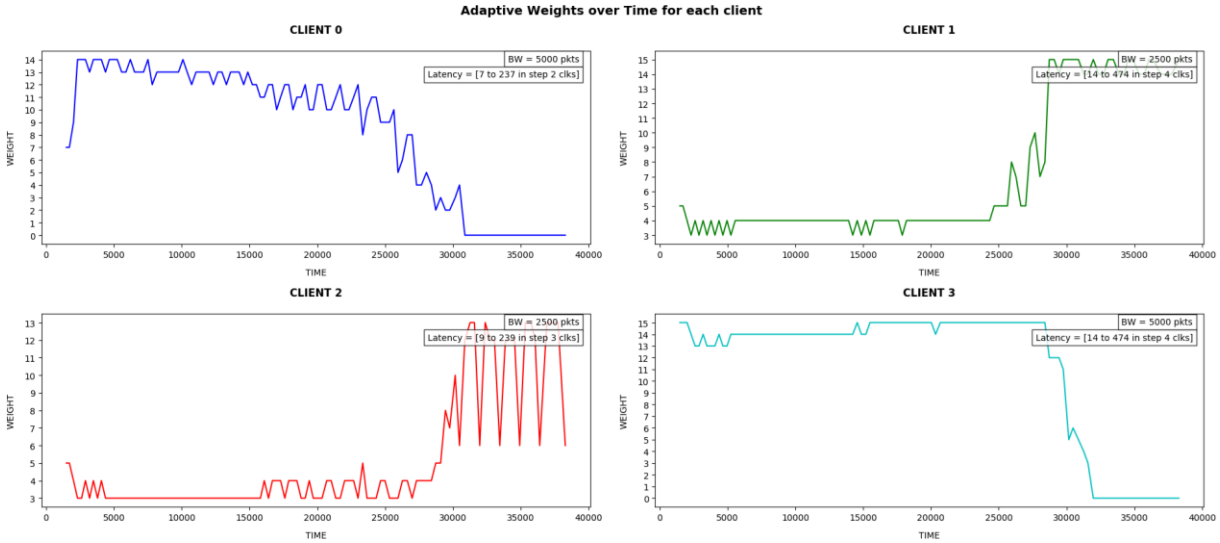


Figure 6. Traffic Generator adapting its weights across clients based on targets per client as well as adhoc system response behavior

*C. Analytical results of matching Silicon Signature Latency targets from Adaptive Bus Modelling Engine*

Below Fig. 7 shows results of how different Masters in Adaptive Bus Modelling Engine are scheduled by the arbiter using adaptive weights generated from history (including error statistics) and learns itself to adapt in terms of future weights calculation to meet the latency targets converging with minimal error margin (<1.25%) w.r.t. silicon dumps:

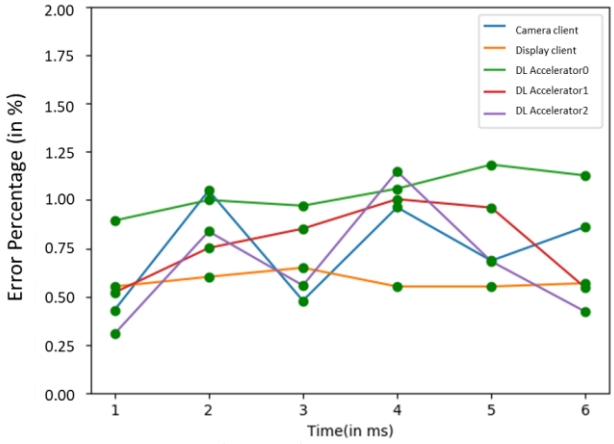


Figure 7. Error % seen across Masters for one of the Real Silicon reproduced issue almost <1.25%

*D. AI based Constraints Solver Performance and Convergence rate comparison*

A summary of comparative analysis between traditional approach (using Cadence XCelium solver) and our approach for constraint solving is captured in Table 2. The comparison is done based on time taken to solve a given set of identical

constraints with use cases having >5000 variables, >15000 constraints and up to 20 levels of dependencies among variables to solve. Furthermore, summary of few KPI comparative analysis between simulation (using Cadence XCelium simulator) and our approach (fully emulated platform on Palladium) are captured in Table 3. The waveform availability is faster since entire design and testbench aspects are on emulation, reducing the need for regular sync between simulation and emulation clock.

TABLE 2
COMPARATIVE PERFORMANCE EVALUATION OF VARIOUS SOLVERS

| Solving type | EDA Solvers | Our approach |
|---|---|---|
| Single Camera IP Solving | 192 sec | 6 sec |
| Partial Camera Chain Solving [ISP Pre Processing 16MP] | 643 sec | 12 sec |
| Full Camera Chain [ISP + Display Processor Chain for Preview] | 981 sec | 13 sec |

TABLE 3
KEY PERFORMANCE INDICATORS (KPI) COMPARISON

| Key Performance Indicators | Traditional approach | Our approach | Comparative metrics |
|---|---|---|---|
| Iterations for silicon signature convergence [ISP + Display Processor Chain for Preview @ FHD 60] | ~34 | ~4 | ~9x faster convergence |
| Latency profile modelling accuracy [Display Processor Preview FHD @ 60fps] | 85% | 98% | >15% gain in accuracy |
| Time taken for debug-ready waveform availability [ISP + Display Processor Chain for Preview @ FHD 60] | 16-18 hours | 20-22 minutes | ~50x faster |

## VI. CONCLUSION

Integrating Agentic AI with architecture awareness into SoC verification accelerates high-fidelity Silicon signature replay. Based on our benchmarking, our proposed solution can offer significant business advantages in terms of time-to-market, including 9x faster corner-case convergence, ~50x faster waveform generation since most aspects of testbench including the Adaptive Traffic generators (modelling the traffic behavior) and Adaptive Bus Modelling Engine (modelling the system latency behavior) are fully synthesizable, ensuring very minimal data exchange between Host and the emulator hardware, also >15% improved memory modeling accuracy compared to state-of-the-art solutions. Selective information flow enabled in the proposed platform enables faster execution in comparison with traditional solvers. Further, it can also reduce reproduction iterations by >80% based on the adaptiveness mechanism embedded in Transaction Generators and Bus Modelling Engine, setting a new benchmark for practical and scalable silicon issue debug methodologies.

In order to extend to other domains, we are planning to make it generic for Auxiliary Processing Unit (XPU) centric designs such as AI accelerators, Discrete graphic processing engines and Multi-core cluster CPU. Furthermore, add more debug features to enhance debug capabilities.

## VII. REFERENCES


[1] P. Pamula, D. P. Gorthy, and A. Alagarsamy. "Verification of SoC Using Advanced Verification Methodology", *Engineering Proceedings*, vol. 34(1), pp. 12, March 2023.

[2] H. Wang, J. Gong, H. Zhang, and Z. Wang. "AI Agentic Programming: A Survey of Techniques, Challenges, and Opportunities", *arXiv preprint arXiv:2508.11126*, 2025.

[3] D. Saha, S. Tarek, H.A. Shaikh, K.T. Hasan, P.S. Nalluri, M.A. Hasan, N. Alam, J. Zhou, S.K. Saha, M. Tehranipoor, F. Farahmandi. "SV-LLM: An Agentic Approach for SoC Security Verification using Large Language Models", *arXiv preprint arXiv:2506.20415*, June 2025.

[4] S. Paria, A. Dasgupta, S. Bhunia. "Towards Automated Verification of IP and COTS: Leveraging LLMs in Pre-and Post-Silicon Stages", In2025 IEEE 43rd VLSI Test Symposium (VTS), IEEE, pp.1-5, April 2025.

[5] Z. Zhang, B. Szekely, P. Gimenes, G. Chadwick, H. McNally, J. Cheng, R. Mullins, and Y. Zhao. Llm4dv: Using large language models for hardware test stimuli generation. In *2025 IEEE 33rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM), IEEE*, pp. 133-137, May 2025.

[5] D. Saha, H.A. Shaikh, S. Tarek, and F. Farahmandi. "ThreatLens: LLM-guided Threat Modeling and Test Plan Generation for Hardware Security Verification", *arXiv preprint arXiv:2505.06821*, May 2025.

[6] Z. Yan, W. Fang, M. Li, S. Liu, Z. Xie, and H. Zhang. "Assertllm: Generating hardware verification assertions from design specifications via multi-llms", *In Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pp. 614-621, Jan 2025.

[7] R. Brummayer, and A. Biere. "Boolector: An efficient SMT solver for bit-vectors and arrays." *In International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 174-177. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[8] J. Wang, L. Zhang, and W. Chen, "An Efficient AXI Traffic Generator for Performance Analysis of Network-on-Chip Architectures," *IEEE Transactions on Very Large Scale Integration Systems*, vol. 28, no. 5, pp. 1256-1269, May 2020.

[9] M. Johnson and S. Thompson, "System and method for AXI-compliant traffic generation," U.S. Patent 9 876 543, Jan. 15, 2018.

[10] Ecco, L.; Ernst, R. Tackling the Bus Turnaround Overhead in Real-Time SDRAM Controllers. IEEE Trans. Comput. 2017, 66, 1961–1974

[11] Ecco, L.; Ernst, R. Improved DRAM Timing Bounds for Real-Time DRAM Controllers with Read/Write Bundling. In Proceedings of the 2015 IEEE Real-Time Systems Symposium, San Antonio, TX, USA, 1–4 December 2015.

[12] S. Agarwal, et al. "gpt-oss-120b & gpt-oss-20b model card." arXiv preprint arXiv:2508.10925, 2025.

[13] A. Yang, et al. "Qwen3 technical report.", arXiv preprint arXiv:2505.09388, 2025.

[14] D. Guo, et al. "DeepSeek-Coder: When the Large Language Model Meets Programming--The Rise of Code Intelligence." arXiv preprint arXiv:2401.14196, 2024ss