An End-to-End Evaluation Framework for NoC IP: Performance Analysis to Verification Support

Chanwoo Song
wade.cw.song@openedges.com
Openedges Technology, Inc.
Seoul, Republic of Korea

Hyun-Gyu Kim
ethan.hg.kim@openedges.com
Openedges Technology, Inc.
Seoul, Republic of Korea

Abstract

High configurability of Network-on-Chip (NoC) in modern System-on-Chip designs creates a major challenge: delivering high-performance interconnects within short time and reliable verification. While automated tools have streamlined RTL generation, performance evaluation has become the main bottleneck. We propose an end-to-end evaluation framework centered on an AXI-compliant, cycle-level traffic generator. The framework employs human-readable JSON workloads to drive identical traffic into both SystemC and RTL models, accelerating exploration at the SystemC level while enabling fine-tuning of design at RTL level. It also introduces a waveform replay mechanism that preserves transaction semantics, maintaining initiator inter-arrival and target service behavior. This enables reproducible workload-driven evaluation and provides practical support for debugging, even when RTL is encrypted. We demonstrate the framework through three workflows: design exploration, cross-abstraction simulation, and waveform replay. Collectively, these capabilities enhance the NoC designer's workflow, reducing evaluation effort and enabling faster, more reliable design closure.

Keywords: Network On Chip, Traffic Generation, RTL - SystemC Cross Simulation, Waveform Replay

1 Introduction

In modern System on Chip (SoC) development, the integration of heterogeneous Intellectual Property (IP) blocks, from general-purpose cores to domain-specific accelerators, has surged, creating new challenges in designing scalable and efficient on-chip communication fabrics. Recent SoCs integrate tens to hundreds of diverse IPs, each potentially demanding distinct performance, bandwidth, and latency characteristics[12, 13] .

This diversity motivates the adoption of irregular network on chip (NoC) topologies that can allocate resources appropriately to the needs of different IPs[11]. Beyond topology, configurability expands across numerous parameters: buffer depths, numbers of multiple outstanding transactions allowed, etc., into a vast design space. This complexity makes one-size-fits-all NoC solutions infeasible. Consequently, NoC IP providers face the necessity of iterative design-release cycles. Each release must be refined to reflect customer workloads. To satisfy customers, vendors must not only reduce the number of iterations but also deliver designs with shorter turnaround time (TAT).

To accelerate this process, we are employing a domain-specific language for network description, the OPENEDGES Architecture Description (OAD) and a compiler toolkit that generates both synthesizable RTL and approximately-timed SystemC[4] models. This enables fast network generation; the bottleneck then shifts to performance evaluation and verification of the generated networks.

Performance evaluation is especially challenging when SoCs run diverse workloads. Addressing this requires modular traffic models and easily configurable performance tests to support rapid, workload-specific exploration. Verification is equally critical: even verified RTL may reveal corner-case errors under customer-specific conditions. Since we release encrypted IP, limited visibility makes debugging particularly difficult.

Existing NoC simulators, such as gem5[5, 10], Garnet[1, 9], BookSim2[7], Noxim[6], and OpenSMART[8] have been instrumental for architectural exploration. However, existing frameworks face two critical limitations. First, generated transactions do not comply with the industry-standard AMBA AXI[2] protocol. While AXI specifies channel-based handshaking for data transfer, current NoC simulators operate at packet or flit granularity. This fundamental mismatch necessitates a legitimate translator to bridge existing simulators with customer-ready RTL or SystemC models. Second, irregular network generation is limited. Whereas existing simulators provide pre-configured regular topologies (meshes, tori, rings), real-world heterogeneous SoCs demand arbitrary irregular topologies. Table 1 compares these frameworks to this work: our approach supports AXI-channel transactions at the subsystem level, handles arbitrary topologies through compilation toolkit.

To overcome these challenges, we have developed an **end-to-end evaluation framework on a cycle-level traffic generator** with four features. First, it provides an extensible and intuitive interface for integrating new traffic models. This enables fine-tuning of customer IP behavior, supporting more accurate and workload-specific exploration. Second, it supports human-readable JSON workload descriptions to

Table 1. Comparison of NoC Simulation Frameworks

Simulator	Cycle-Level	Format	SystemC Sim.	RTL Sim.	Irregular Topologies	Traffic
gem5[5, 10]	○	Packet	Needs translation	Needs translation	×	System-level
Garnet[1, 9]	○	Flit			×	System-level
BookSim2[7]	○	Flit			×	Synthetic
Noxim[6]	○	Flit			×	Synthetic
OpenSMART[8]	○	Flit		△ (limited)	△ (limited)	Synthetic
This work	○	AXI	○	○	○	Subsystem-level[1)]

[1)] System-level integration in progress

streamline scenario design and iteration. This allows designers to generate diverse workloads quickly and to construct a more refined parameter space, ensuring that customer requirements are clearly reflected. Third, it enables cross-abstraction simulation with SystemC model, offering significant speedup in evaluation compared to RTL simulation. Finally, the framework provides a waveform replay mechanism that preserves transaction semantics including initiator inter-arrival timing and target service latency. This mechanism allows customer's real workload-based performance tests to be reapplied across alternative networks while keeping endpoint behavior fixed, thereby simplifying performance studies under realistic workloads. In addition, when replayed on the same network, it reconstructs internal behavior in a *cycle-exact* manner, enabling debug localization from boundary waveforms alone and thus accelerating the verification process.

The remainder of the paper is organized as follows: Section 2 describes the architecture and design philosophy of the framework. Section 3 demonstrates three representative workflows: design exploration (Section 3.1), RTL–SystemC cross abstraction simulation using identical traffic (Section 3.2), and waveform replay for reappliance of customer's evaluation workloads (Section 3.3), highlighting wide coverage of this framework's functionality. Section 4 concludes the paper by wrapping up the main contributions and outlining future directions.

2 Architecture

This section presents the design philosophy and end-to-end architecture of the traffic generator (TG), core of our evaluation framework. Figure 1 describes the architecture with callouts (1)–(6) referenced below.

Thin, extensible core API interface (1). The TG is constructed on two base classes, `AbstractInitiator` and `AbstractTarget`. Each exposes a compact set of virtual functions that the Design under test (DUT) invokes once per clock for the connected port. This per-cycle function call lets the TG synthesize cycle-level requests and responses while keeping the DUT binding minimal. Because only this interface is bound to the DUT, users are free to implement arbitrarily complex behaviors. A user can implement a complex call chain behind the virtual functions to model sophisticated behavior or an event queue for a more efficient simulation with an event-driven manner. We currently provide initiator-side models (benchmark-mocking generators, synthetic patterns, and trace replayers) and target-side models (synthetic responders and trace replayers) and continuously expanding models that imitate IP behaviors.

Declarative workloads via JSON (2). Workloads are specified in a concise, human-readable JSON schema. Each top-level entry describes a TG instance with fields such as: `role`, `type`, `params`. This declarative layer decouples experiment design from compilation. It encourages sweeping large spaces (e.g., allowed outstanding transactions, burst distribution, address patterns) without touching C++ code, and it makes runs easy to version, compare and reproduce.

Runtime factory (3). A lightweight factory resolves each JSON entry to a concrete class at runtime. The `type` field selects the concrete implementation; the corresponding `params` block is validated and used to initialize the model. This compile-once/run-many structure is especially advantageous for designs with numerous initiators and targets: users link the TG once, then scale experiments by editing JSON only. In practice, we provide schema validation to catch malformed `params` early and to emit defaults deterministically.

Transaction-to-interface bridges (4), (5). The TG produces transactions in an AXI-compliant, C++ class view; two bridges translate this view to the DUT's interfaces appropriately:

RTL bridge (DPI-C). It converts each transaction into pin-level AXI signals across the five channels (AW, W, B, AR, R) and follows AXI handshake semantics. Timing fields (e.g., ready/valid, burst boundaries by LAST signals, and inter-transaction intervals) are preserved so that RTL sees the exact intended stimulus.

TLM bridge. It wraps the same transaction into an AMBA TLM-2.0[3] AXI transaction payload. This enables cross abstraction experiments: an RTL and a SystemC model can be simulated with equivalent traffic (addresses, burst lengths, IDs, and other control signals) with identical cycle-level timing.

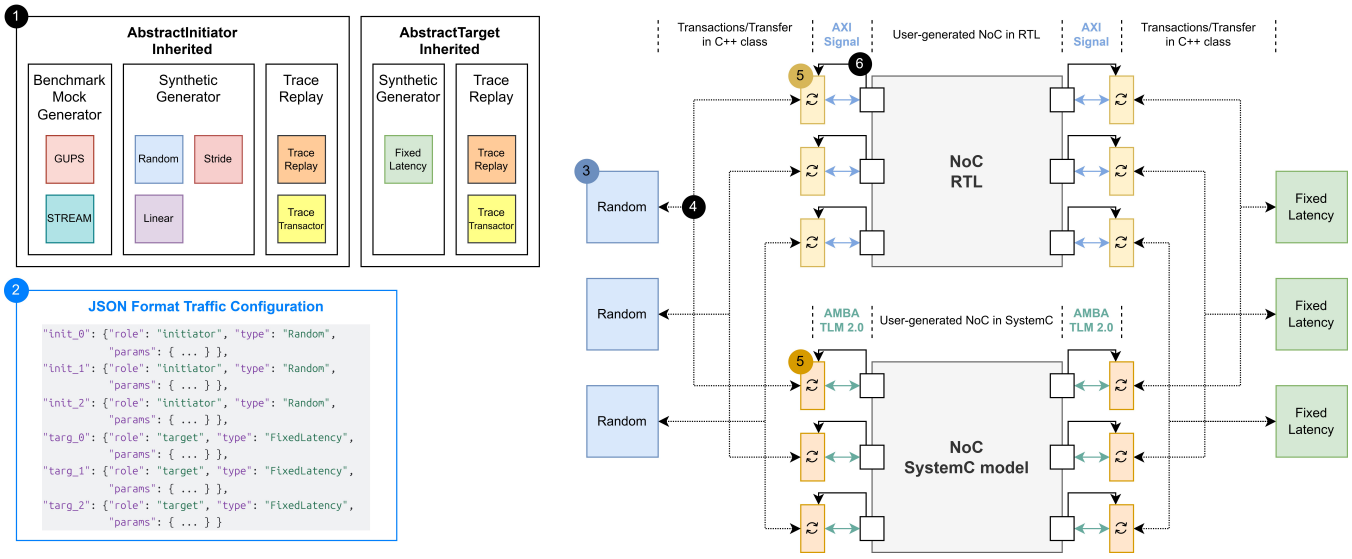

Figure 1. TG architecture. (1) initiator side: AbstractInitiator family with benchmark-mock generators, synthetic generators, and trace replay/transactor. target side: AbstractTarget family with a synthetic responder and trace replay/transactor. (2) JSON traffic configuration declaring role/type/params. (3) A generator instance (e.g., Random). (4) Per-cycle callback path producing C++ transaction objects. (5) Per-port bridges translating transactions to pin-level signal for RTL and to AMBA TLM-2.0 for SystemC. (6) Time source: each port's clock drives its bridge.

Both bridges provide an infinite FIFO buffer at the interface boundary. If the network saturates or the DUT backpressures, requests can queue up in the bridge. The TG records latency from request creation (when the model issues it) to drive/handshaked time. This yields accurate end-to-end latency accounting even when generation runs ahead of interface availability. In addition to precise record fo network latency, a user can define a maximum number of allowed buffered transactions and determine whether the network can provide sufficient performance for a given workload.

Time source and multi-clock domain simulation (6). The top ports of the DUT act as the time source of connected traffic models. Each connected bridge drives its per-port call chain on that port's clock edge—positive edge for RTL and (by convention) negative edge for the SystemC model. This design frees users from manually synchronizing model clocks to port frequencies. Different TG instances naturally run at the DUT's respective port rates, making multi-clock domain traffic generation simple and legitimate.

Statistics and tracing. Every bridge owns a collector module following the same design pattern as generators: a small fixed virtual interface (via an `AbstractCollector`) with pluggable implementations. We provide (i) a full-trace collector that logs every transaction and handshake for high-fidelity analysis and fine-tuning, and (ii) a summary collector that aggregates throughput, latency distributions, backpressure ratios, FIFO occupancy, and drops (if any) for fast, flexible space exploration. Users select the collector per instance in JSON, enabling a mix of deep tracing on a few important ports and lightweight counters elsewhere within the same run.

3 Evaluation

This section demonstrates three representative workflows that leverage our framework's functionality. First, we conduct network evaluation on a fixed system configuration, which is essential for narrowing candidates and identifying the most promising topologies (Section 3.1). Next, we present RTL–SystemC cross-abstraction simulation using identical traffic, which takes advantage of SystemC's lighter execution to enable more efficient performance assessment while preserving cycle-level behavior (Section 3.2). Finally, we showcase waveform replay, which preserves transaction semantics such as initiator inter-arrival and target service time. This mechanism allows customer workloads to be reapplied consistently, making it easier to reproduce performance tests and support design exploration under realistic traffic conditions (Section 3.3). Collectively, these workflows highlight the framework's ability to reduce evaluation turnaround, facilitate workload-driven exploration, and provide practical support for the design and verification of NoC interconnects.

3.1 Design Exploration

After an initial design-space exploration at higher abstraction, detailed evaluation of selected promising candidate networks is required. Three key objectives for the evaluation stage are: (i) ensuring that actual traffic demands can be processed in low-level models such as SystemC or RTL,

Table 2. Initiator and target configuration

Initiator	Data Width (bit)	Burst Length (avg/peak)	R/W Ratio	Allowed MOs (R/W)	Bandwidth (avg/peak, MB/s)
CPU0 / CPU1	256	4 / 4	6 / 4	32 / 32	17500 / 6000
GPU0 / GPU1 / GPU2 / GPU3	256	4 / 4	5 / 5	32 / 64	22500 / 6000
NPU0 / NPU1	512	8 / 8	5 / 5	256 / 256	40000 / 25000
Target					
DDR0 / DDR1 / DDR2 / DDR3	256	-	-	40 / 40	-

(ii) confirming that a candidate satisfies customer-defined hard constraints such as bandwidth requirement, and (iii) capturing cost-performance tradeoff and select the optimal network. This experiment demonstrates our methodology to proceed the evaluation stage.

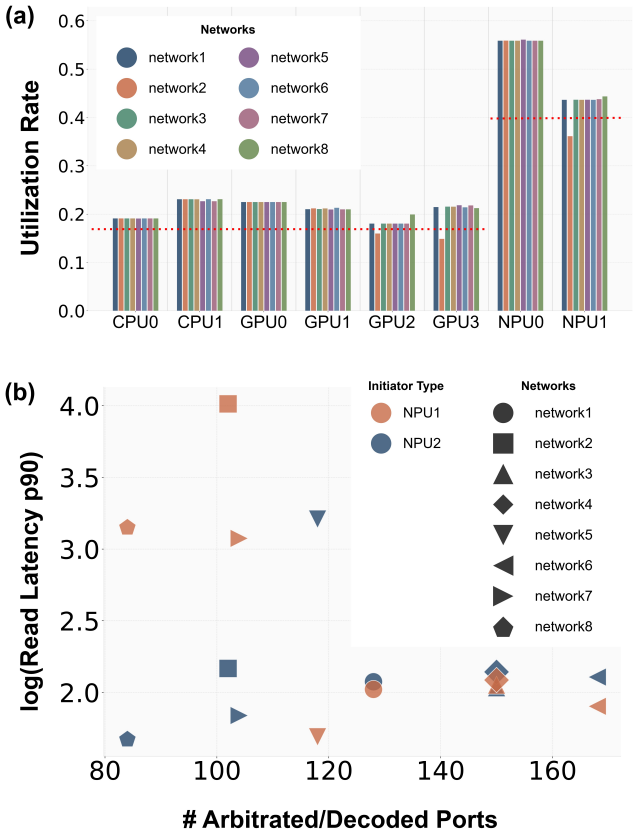

Figure 2. Exploration workflow on candidate networks. (a) Utilization rates for each initiator across candidates. The red dashed line marks a hard utilization constraint. (b) Pareto plot of candidates, where the x-axis denotes cost proxy (arbitrated/decoded ports) and the y-axis represents evaluation metric (log of p90 read latency).

Experimental Design. The system configuration is summarized in Table 1. It models a heterogeneous SoC with initiators including two CPUs, four GPUs, and two NPUs, and four DDR memory targets. Each initiator entry specifies the data width, average and peak burst lengths, read/write ratios, allowed multiple outstanding (MO) transactions, and required bandwidth. These parameters collectively define both the traffic injection behavior and memory demand of each IP. For instance, NPUs operate with a wider data width (512 bit) and significantly higher outstanding transactions (256/256), reflecting their throughput-critical role.

Candidate networks were prepared by randomly generating the OADs. We have synthetically generated traffic based on the profile described in Table 1, and simulated all networks with identical traffic. Firstly, we focused whether candidate networks satisfy the bandwidth requirement (Figure 2a). Then, under the assumption that a customer's primary concern is service time of NPU traffic, we captured 90th percentile of read transaction latency at the NPU initiator side as our metric. The cost proxy was computed as the number of arbitrated and decoded ports, directly derived from the OAD of corresponding network. This reflects structural complexity of the network and the area of the design implicitly. We plotted the tradeoff space as a Pareto plot (Figure 2b), with cost proxy on the x-axis and the evaluation metric on the y-axis.

Result. Figure 2a presents the utilization rate of each initiator across candidate networks. The red dashed line represents the utilization requirement to be met. As depicted, network 2 can be excluded from our candidate as it failed to satisfy the constraint. Figure 2b presents the cost-performance relation across candidate networks. Networks with lower cost proxies (< 120 ports) exhibit optimal latency for one NPU initiator while providing unacceptable latency for another, indicating that network resource was not provided in a balanced manner. On the other hand, networks with higher cost proxies (>120 ports) exhibit acceptable latency for both initiators. Based on the Pareto plot, it seems reasonable to select network 1 among others, as it provides optimal metric with relatively low cost.

Implication. The experiment illustrates how the proposed framework enables a comparative evaluation of NoC

designs. The framework allows to formulate lambda functions for cost proxies and performance metrics with various visualization utils, therefore users can flexibly configure Pareto analysis across arbitrary design objectives.

3.2 RTL-SystemC Cross Abstraction Simulation

Executing simulations within a short time enables the exploration of a broader set of design alternatives or the faster TAT. To satisfy this demand, our framework also supports performance evaluation using SystemC models, because SystemC provides a significant speedup while still preserving cycle-level behavior of RTL. This experiment highlights the framework's ability to drive identical traffic into both RTL and SystemC models at cycle-level, ensuring consistent evaluation across abstraction levels.

Experiment design. A single OAD was compiled into both RTL and SystemC models using our compilation toolkit (Figure 3a). These two models represent functionally equivalent networks. Simulation for both models was then executed under the same traffic configuration. For each transaction, all control fields and timing information (e.g., cycle when requested and completed) were recorded.

Each recorded transaction was matched across the two models by its control fields. For corresponding transactions, latency values were computed independently from the RTL and SystemC runs, and the difference was calculated. These latency differences were then aggregated and plotted as histograms (Figure 3b for read and Figure 3c for write).

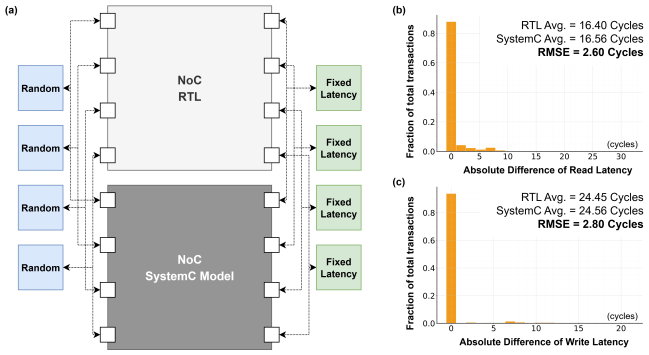

Figure 3. RTL-SystemC cross-abstraction simulation. (a) RTL run driven by random generators and fixed-latency targets; traces recorded. (b) SystemC run driven by text-trace replayers at initiators and targets. (c) Histogram of per-transaction read-latency difference (SystemC-RTL; fraction of transactions). (d) Histogram of write-latency difference.

Result. Under identical stimulus, the average read latency was 16.40 cycles in RTL and 16.56 cycles in SystemC, corresponding to an RMSE of 2.60 cycles. Write latency averaged 24.45 cycles in RTL and 24.56 cycles in SystemC, with an RMSE of 2.80 cycles. In both cases, the latency-difference histograms exhibit sharp peaks at zero, confirming close alignment between the two models and demonstrating acceptable cross-model correlation.

Implication. These results highlight two important implications. First, by injecting identical traffic at cycle granularity into both RTL and SystemC models, the framework enables faster performance evaluation. SystemC models of different network candidates can be assessed using the same methodology described in the previous section. Second, the ability to collect and compare transaction-level traces provides a practical means to localize discrepancies between the two models. This not only supports validation but also guides fine-tuning of the SystemC model, ensuring the model as a faithful surrogate for RTL.

3.3 Waveform Replay

After the integration of an NoC design into a customer's SoC, the most practical way to explore alternatives is to evaluate networks under the same workloads used in customer's real workload-based evaluation. Simply replaying raw timing of customer-provided waveform would break transaction semantics; for example, generating responses without valid requests arrival. Therefore, initiator inter-arrival and target service behavior must be preserved. This experiment showcases our framework's waveform replay mechanism, enabling consistent reuse of customer workloads.

Experiment design. A single OAD was compiled into RTL and a simulation was proceeded with random generators (Figure 4a). During execution, we captured a Value Change Dump (VCD) of the boundary signals and transaction-level traces at initiator and target bridges. As in previous experiment, all control fields and timing information were recorded. We then replayed the captured VCD using the framework's built-in VCD parser and initiator/target VCD transactors (Figure 4b), again recording transaction-level traces. Finally, we compared the latencies measured at the initiator and target interfaces between the original and replay runs.

Result. The scatter plots in Figure 4c–f exhibit a clear y = x behavior for both reads and writes, across both initiator and target sides: every point lies on the diagonal, confirming that each transaction in the replay completed with identical latency to its counterpart in the original run.

Implication. This experiment confirms that by preserving transaction semantics, namely initiator inter-arrival timing and target service latency, the framework can faithfully reproduce customer workloads in replay. Such fidelity ensures that endpoint behavior is maintained, enabling reliable reapplication of performance tests under realistic traffic conditions. Moreover, as the replay reproduces *cycle-exact* behavior on the same network, it can also serve as an effective tool for verification support: this capability enables engineers to pinpoint the source of bugs more effectively, thereby accelerating customer-side verification.

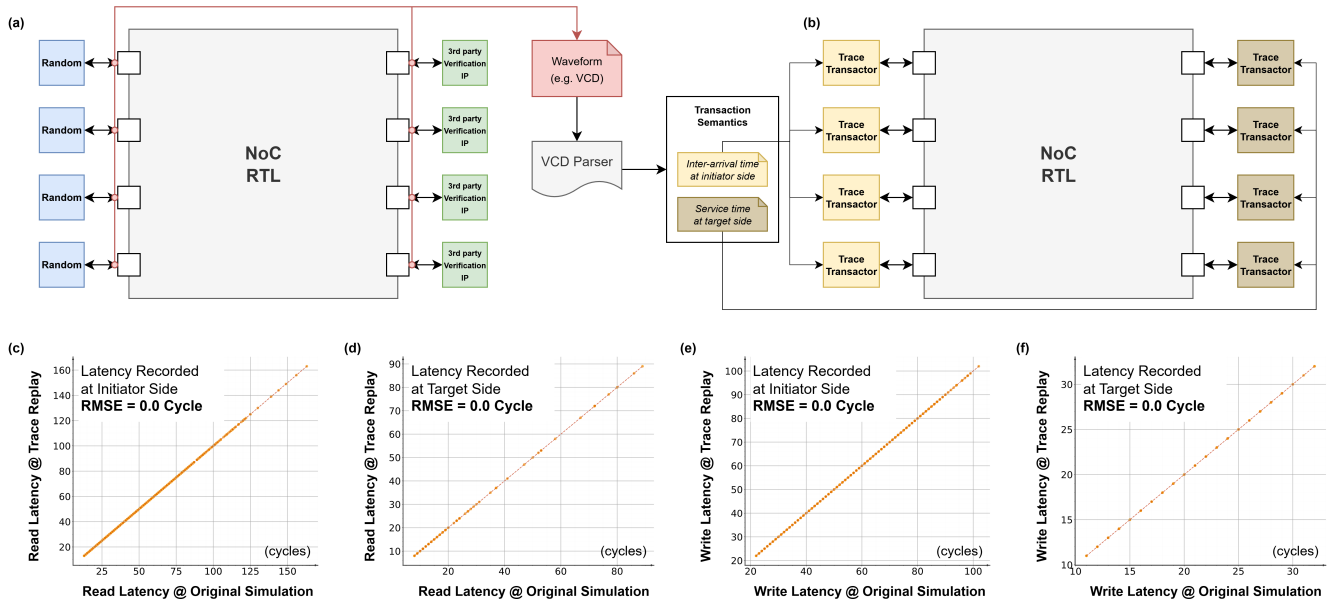

Figure 4. Waveform replay. (a) RTL experiment: random traffic driven into the network, boundary waveform and transaction traces captured. (b) Replay experiment: original waveform parsed and injected back into the network via VCD transactors, reconstructing transaction semantics. (c)–(f) Scatter plots comparing original vs. replay latencies: (c) read at initiator, (d) read at target, (e) write at initiator, (f) write at targe.

4 Conclusion

This work has presented an end-to-end evaluation framework built on a cycle-level traffic generator to address two persistent challenges in NoC IP design: efficient performance exploration and verification support. The framework integrates four key capabilities. First, it provides an extensible and intuitive interface that allows new traffic models to be incorporated with minimal effort. Second, it supports human-readable workload specifications in JSON, enabling rapid construction and iteration of diverse evaluation scenarios. Third, the framework can drive cross-abstraction simulation by driving identical traffic into both RTL and SystemC models. Once validated, the SystemC estimator can stand in for RTL, making broader design space exploration efficient while retaining cycle-level fidelity. Finally, the framework introduces a waveform replay mechanism that preserves transaction semantics, including initiator inter-arrival timing and target service latency. This feature allows customer workloads to be reapplied consistently, enabling meaningful and reproducible performance studies under realistic traffic conditions. At the same time, because replay faithfully reconstruct *cycle-exact* behavior on the same network, it provides practical verification support by allowing engineers to localize bugs from boundary traces and accelerate the debugging process.

Together, these capabilities form a practical methodology to navigate the complex design space of heterogeneous SoCs. The experiments demonstrated in this paper, candidate design exploration, SystemC/RTL cross-abstraction simulation, and waveform replay, illustrate how the framework shortens overall TAT by supporting and facilitating designer workflow. Looking ahead, the framework can be extended toward data-driven synthesis of irregular NoC topologies by leveraging its modular APIs for new workload models and design objectives. It thus provides immediate utility for current SoC designs while offering a foundation for future workload-driven interconnect methodologies.

References

[1] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K Jha. 2009. GARNET: A detailed on-chip network model inside a full-system simulator. In *2009 IEEE international symposium on performance analysis of systems and software*. IEEE, 33–42.

[2] Arm Limited. 2023. *AMBA® AXI Protocol Specification*. Specification ARM IHI 0022. Arm Limited, Cambridge, UK. Non-confidential; released Sep 29, 2023.

[3] Arm Limited. 2023. *AMBA® TLM 2.0 Library Reference Manual*. Reference Manual 101459_02_en. Arm Limited, Cambridge, UK. Issue 02; Non-Confidential; Second release on June 1, 2023.

[4] IEEE Standards Association et al. 2012. IEEE Standard for standard SystemC language reference manual. *IEEE Computer Society* (2012).

[5] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R Hower, Tushar Krishna, Somayeh Sardashti, et al. 2011. The gem5 simulator. *ACM SIGARCH computer architecture news* 39, 2 (2011), 1–7.

[6] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. 2015. Noxim: An open, extensible and cycle-accurate network on chip simulator. In *2015 IEEE 26th international conference on application-specific systems, architectures and processors*

(ASAP). IEEE, 162–163.

[7] Nan Jiang, Daniel U Becker, George Michelogiannakis, James Balfour, Brian Towles, David E Shaw, John Kim, and William J Dally. 2013. A detailed and flexible cycle-accurate network-on-chip simulator. In *2013 IEEE international symposium on performance analysis of systems and software (ISPASS)*. IEEE, 86–96.

[8] Hyoukjun Kwon and Tushar Krishna. 2017. OpenSMART: Single-cycle multi-hop NoC generator in BSV and Chisel. In *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. IEEE, 195–204.

[9] Ren-Min Li, Chung-Ta King, and Bhaskar Das. 2016. Extending Gem5-garnet for efficient and accurate trace-driven NoC simulation. In *Proceedings of the 9th International Workshop on Network on Chip Architectures*. 3–8.

[10] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. 2020. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152* (2020).

[11] Christian Neeb and Norbert Wehn. 2008. Designing efficient irregular networks for heterogeneous systems-on-chip. *Journal of Systems architecture* 54, 3-4 (2008), 384–396.

[12] Semiconductor Research Corporation. 2023. *Microelectronics and Advanced Packaging Technologies (MAPT) Roadmap*. https://srcmapt.org/

[13] Hansika Weerasena, Xiaoguo Jia, and Prabhat Mishra. 2025. Topology-aware Detection and Localization of Distributed Denial-of-Service Attacks in Network-on-Chips. *arXiv preprint arXiv:2505.14898* (2025).