PANE: Pluggable Asynchronous Network-on-Chip Simulator

SNEHA N. VED, SARABJEET SINGH, and JOYCEE MEKIE, Indian Institute of Technology Gandhinagar

Communication between different IP cores in MPSoCs and HMPs often results in clock domain crossing. Asynchronous network on chip (NoC) support communication in such heterogeneous set-ups. While there are a large number of tools to model NoCs for synchronous systems, there is very limited tool support to model communication for multi-clock domain NoCs and analyse them. In this article, we propose the **P**luggable **A**synchronous **NE**twork on Chip (PANE) simulator, which allows system-level simulation of asynchronous network on chip (NoC). PANE allows design space exploration of synchronous, asynchronous, and mixed synchronous-asynchronous(heterogeneous) NoC for various system-level NoC parameters such as packet latencies, throughput, network saturation point and power analysis. PANE supports a large range of NoC configurations—routing algorithms, topologies, network sizes, and so on—for both synthetic and real traffic patterns. We demonstrate the application of PANE by using synchronous routers, asynchronous routers, and a mix of asynchronous and synchronous routers. One of the key advantages of PANE is that it allows a seamless transition from synchronous to asynchronous NoC simulators while keeping pace with the developments in synchronous NoC tools as they can be integrated with PANE.

CCS Concepts: • **Hardware → Network on chip**;

Additional Key Words and Phrases: NoC, asynchronous simulator, heterogeneous systems

ACM Reference format:
Sneha N. Ved, Sarabjeet Singh, and Joycee Mekie. 2019. PANE: Pluggable Asynchronous Network-on-Chip Simulator. *J. Emerg. Technol. Comput. Syst.* 15, 1, Article 7 (January 2019), 27 pages.
https://doi.org/10.1145/3241051

1 INTRODUCTION

In the past five years, Nvidia has launched several large heterogeneous chips in the Tegra family, starting with Tegra 2 [7], Tegra Parker [36], and, the most recent, Tegra Xavier [8], announced in September 2016. Heterogeneous architectures like that of the Nvidia Tegra Xavier pack processing elements of different kinds together on the same chip. These heterogeneous cores can be clocked at different clock frequencies. The processing elements could be CPU cores, GPU cores, on-chip memory and some special function cores like the encryption-decryption engine [35].

Heterogeneous multicores often require an on-chip interconnect that supports communication across clock-domains present on the chip [29, 31, 46]. In a homogeneous set-up, since the clock frequencies are the same, the on-chip network is also called a synchronous network. However, when

Authors' addresses: S. N. Ved, S. Singh, and J. Mekie, Department of Electrical Engineering, Indian Institute of Technology Gandhinagar, Gandhinagar, Gujarat-382355, India; emails: {sneha.ved, sarabjeet.singh, joycee}@iitgn.ac.in.

1550-4832/2019/01-ART7 $15.00
https://doi.org/10.1145/3241051

the on-chip network connects heterogeneous cores, it is also called the asynchronous network, as the cores could possibly be clocked at different clock phases or frequencies.

Several system-level simulators [11, 27] for synchronous on-chip networks, discussed in Section 3, are used to assess the high-level performance of the on-chip networks, especially for benchmark applications. Support for simulation of benchmarks such as PARSEC [16] on the modeled NoC is essential as it allows the NoC designers to provide fair and complete evaluation of different NoCs. As benchmarks represent realistic NoC usage scenarios, they allow more accurate assessment of the NoC performance in real-world situations. This assessment of synchronous NoCs is carried out by modeling them in the simulators using high-level languages such as C++ and SystemC in the simulators. This allows relatively faster prototyping of the NoC behavior along with a high-level comparison of the performances and power of different NoC architectures. Many of these simulators are shipped with built-in models for popular NoC schemes proposed in the literature earlier. This includes routing, allocation and arbitration mechanisms, topologies, and router microarchitectures. However, unlike synchronous NoCs, we have not come across a mature simulator for asynchronous and mixed synchronous-asynchronous NoCs that are functionally comparable to the simulators available for synchronous NoCs and carry out system-level simulations with similar benchmark support. As in case of synchronous NoC simulators, it is ideal that the simulator for asynchronous and mixed synchronous-asynchronous systems supports modeling and analysis of different topological connections in the NoC; different allocation, arbitration, and routing schemes; and different router microarchitectures and benchmark support.

One of the major advantages of the asynchronous design is the average case performance offered by data dependent pipeline stage delay vis-a-vis the synchronous design worst-case pipeline stage delay [34]. The simulator for asynchronous NoCs should also be able to model data dependent variability at different handshake units and arrival-time dependent arbitration variability at the router crossbar. These variabilities are inherent to asynchronous circuit designs and need to be accounted for in the high-level simulator designed for asynchronous NoCs.

In this article, we propose a pluggable asynchronous network-on-chip (PANE) simulator that supports modeling and analysis of synchronous, asynchronous and mixed synchronous-asynchronous NoCs. PANE is an event-driven synchronous, asynchronous, mixed synchronous-asynchronous and multi-clock domain NoC simulator. Event-modeling not only supports asynchronous communication but also allows PANE to model variability in synchronous routers and NoCs. PANE models handshake-based asynchronous timing-control paths. It allows easy modeling for a variety of NoC designs with different topologies, routing algorithms, allocation and arbitration mechanisms, buffering schemes and workloads. PANE's design allows for simulation of separate yet interacting implementations of the data, routing-control, and timing-control paths of the NoC, partly at the cost of simulation time. Asynchronous timing-control-path modeled in PANE supports bundled data as well as quasi delay-insensitive (QDI) asynchronous data-transfer protocols [37]. PANE builds on the implementations of data and routing control paths in synchronous NoC while providing mechanisms to implement asynchronous handshake. This allows the PANE framework to stay at par with the synchronous NoC design tools available. In the presented work, we implement data and routing control path by modifying BookSim2 [1, 27].

The key contributions of this work are as follows:

(1) PANE is the first modeling and system analysis tool for asynchronous, synchronous and mixed synchronous-asynchronous (heterogeneous) NoC that is at par with the synchronous NoC analysis tools in terms of functionality. PANE not only models handshake-based asynchronous communication between the NoC components but also models data-dependent as well as arrival-time dependent arbitration delays in asynchronous circuits.

These delay parameters are user-configured, randomized, and bounded, allowing more accurate and realistic modeling of asynchronous circuits.

(2) PANE has been designed to model the system effects of variability in synchronous NoCs. With the rise of variability-related issues in circuit design, PANE provides mechanisms to model and quantify the system-level impact of this.

(3) PANE supports (a) comprehensive analysis across different NoC topologies, router designs, routing algorithms, flit size, buffer sizes, and buffering schemes and (b) reports the network performance parameters such as network latency, network throughput, flit and packet latencies with varying injection rates, traffic patterns, and workloads.

(4) PANE allows use of existing NoC simulators for modeling data path. As an augmentation to existing NoC tools, PANE can easily stay at par with its synchronous counterparts in terms of functionality.

(5) PANE has been validated for functional correctness against BookSim2 that in turn has been validated against its Resistor-Transistor Logic (RTL) implementation.

PANE simulator has been tested on up to 32×32 NoC with both synchronous and asynchronous routers for PARSEC benchmark suite [16] and different synthetic benchmarks with injection rates varying from 0.05-0.95 flits/router/cycle. PANE currently supports different topologies (mesh, torus etc.), routing algorithms (DOR-XY), buffering, allocation and arbitration schemes, packet sizes, injection rates and real benchmark applications and synthetic traffic patterns.

Availability: PANE will be made available as public domain.

The rest of the article is organized as follows: Section 2 gives the motivation to develop PANE. In Section 3, we discuss other works in literature related to this work. In Section 4, we give the design and operation details of PANE. In Section 5, we discuss how PANE is validated against BookSim2 and the RTL for asynchronous NoC and then verify it for correctness, deadlocks and livelocks. In Section 6, we present a few use cases for PANE simulator. We conclude the presentation of this work in Section 7.

2 MOTIVATION

In synchronous multicore systems, communication within the synchronous routers and across the links (between the NoC routers) happens at clock events. Several simulators to model and analyze synchronous NoCs have been proposed in the literature, as discussed in Section 3. These simulators allow analysis of the system-level impact of design choices made for NoCs. The available synchronous NoC simulators model cycle based behavior of the NoC.

Due to third-party IP core based designs, the need for finer power control and better economics of computing [25], systems are increasingly being designed as globally asynchronous locally synchronous (GALS) [13] systems. These systems are organized as a collection of synchronous clock domains that communicate asynchronously to each other. Here, while the routers within a clock domain are synchronous, the ones communicating across the clock domains need to support asynchronous (multi-clock domain) communication.

Asynchronous designs are gaining popularity as the complexity and overheads of clock distribution network increase [13]. Several routers, such as in References [18, 22, 39], are built to operate asynchronously, i.e., without a clocked control. Instead of clocks, asynchronous pipelines are realized using handshake protocols. PANE is the first simulator to model, simulate, and analyze the behavior of GALS and asynchronous NoCs in addition to modeling synchronous NoCs, as shown in Figure 1.

As compared to synchronous NoC simulators, design of asynchronous NoC simulator primarily differ in the following aspects:

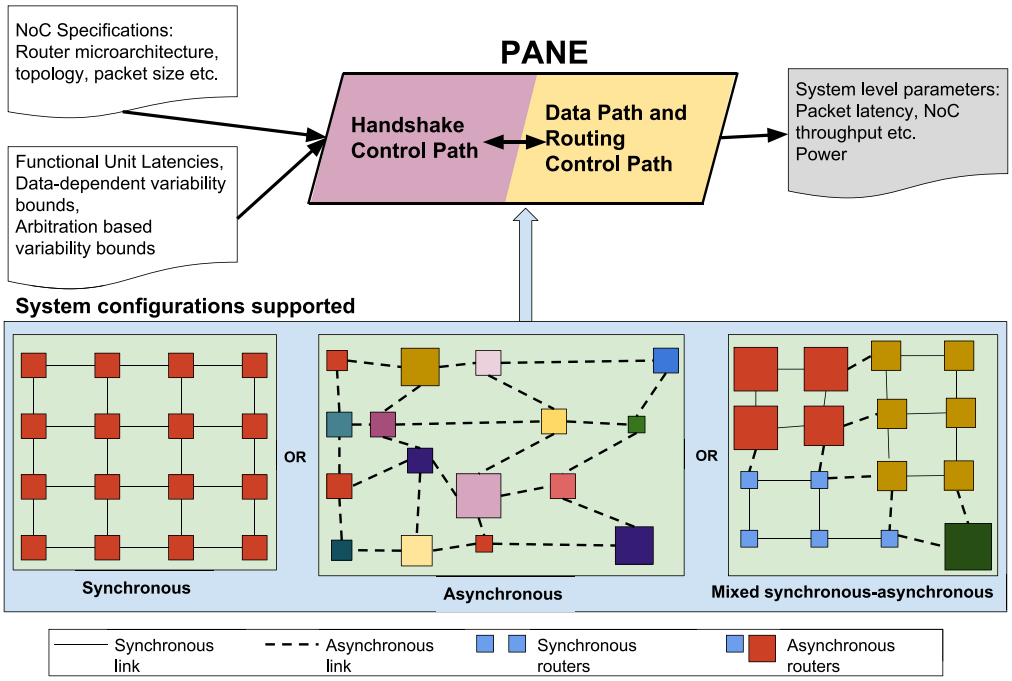


Fig. 1. Schematic of interaction between the 4 × 4 mesh asynchronous NoC in synchronous, asynchronous, and mixed synchronous-asynchronous configurations with **PANE**.

- Notion of time: clocked versus absolute time,
- Data dependent delays and
- Arrival time dependent arbitration delays.

Progress of data in a synchronous pipeline is governed by clock events. This periodic synchronous behavior can be modeled in a simulator using iterative evaluation of functional units, where each iteration represents a clock event. Since the delays of all the synchronous functional units are measured as the number of clock events, each functional unit is designed to complete its actions in a fixed and finite number of clock cycles. Like the other functional units in synchronous systems, arbiters sample the input requests at a clock event and produce output grants at a subsequent clock event. This delay between sampling the input requests and producing the output grants is fixed finite and irrespective of the order and time of arrival of the input requests. Therefore, in a synchronous system simulator, the simulator iterates through the models of all the functional units, including the arbiters, evaluating them at the specified clock event. Synchronous NoC simulators continue evaluation of inputs in the functional units for the specified number of clock cycles (i.e. iterations) or until a specified input trace is exhausted.

On the contrary, asynchronous pipelines are realized using handshakes. The data moves into a functional unit of the pipeline when the functional unit has a "token" available. Token indicates the availability of the functional unit to process new inputs. Absence of a token therefore indicates that the functional unit is busy processing an earlier input. Thus, in asynchronous systems, the time is measured as wall-clock (real) time instead of clock cycles, making them event-based systems. Unlike synchronous systems that are periodic, asynchronous systems require event-sensitive simulators that can capture and model systemwide events. In other words, time is quantized as clock cycles in synchronous systems, which is not the case with asynchronous systems.

In our model of asynchronous handshakes in PANE, we divide time into smallest acceptable resolution called *epoch*. Notion of time in PANE is discussed in detail in Section 4.5. The time taken by the same functional units might also differ based on the inputs applied to the it. This is called data-dependent delays. Therefore, it is desirable that the asynchronous system-simulators account for these data-dependent delay variations in their models. Additionally, unlike in synchronous systems, asynchronous arbiters sample the inputs as they arrive. If the input requests in asynchronous arbiters arrive too close to each other (clash), then the arbiter can take an indefinitely long time to resolve. This behavior, unlike in synchronous systems, is indeterministic and also needs to be modeled. PANE captures these key features of asynchronous system design to simulate accurate asynchronous NoCs.

3 RELATED WORK

In this section, we survey the relevant literature. We report the literature in the area of simulation of RTL for synchronous and asynchronous circuits, some asynchronous NoC architectures and the available system-level architectural simulators for multicores NoCs. The proposed synchronous asynchronous NoC simulator PANE is a high-level architectural simulator for synchronous, asynchronous and mixed synchronous-asynchronous NoCs.

Resistor-Transistor Logic (RTL) Simulators

Several simulators such as SPICE [33], Spectre [28], Incisive [23], IVerilog [4, 45], and so on, are used for circuit test and design. However, high-level simulators allow simulation of the high-level description of the system. In these, the system description is often modular and the time-to-model is much shorter than the time to develop an RTL. Modular design allows easier design space exploration. Also, since these simulators indicate the impact of a design choice on the overall system, they assist in making informed architectural design choices. However, these high-level system simulators only approximately estimate the power, area and performance of the architectural module.

Balsa [12] and PARBREEZE [40] provide a framework for description of asynchronous circuits. Like RTL simulators, they provide accurate estimates of power, area, and performance of the implemented design. However, like RTL simulators,these tools are restricted by the computation complexity and time-to-model.

Asynchronous NoCs

Asynchronous routers such as MANGO [18], QNOC [19, 22], and ALPIN [39] have been proposed. In most of these works, RTL synthesis results have been reported to compare the performances of these routers. However, the impact of the router designs on the NoC performance cannot be easily compared without a high-level NoC simulator. For instance, in synchronous NoCs, consider CHIPPER [24]. The authors report that the CHIPPER router has a critical path of 1.9ns while another router BLESS [32] has a critical path of 2.68ns (Table 2 in Reference [24]). Therefore, NoC using CHIPPER routers would ideally be faster, i.e., the packets should experience lower packet latency in a CHIPPER based network. However, in Figure 6(a) of Reference [24], it is clear that the packets in a BLESS based NoC see lower packet latency across all injection rates for uniform random traffic. Therefore, a platform that allows overall system-level is required for a holistic evaluation of NoCs.

System and NoC Simulators

GEM5 [2, 17] and Multi2Sim [5, 41] are full-system simulators used for core and memory-organization analysis on single core or multi-core systems. GEM5 supports *Ruby* and GARNET [11] synchronous NoC simulators. GARNET is closely coupled with the memory subsystem and can simulate real traffic conditions based on the memory transactions generated in the system. For

Table 1. Comparison of HNOCS and PANE

Feature	HNOCS	PANE (Proposed)
Data dependent delay	Not modeled	Modeled
Arbitration delay	Not modeled	Modeled
Simulation Time	29s	32s
Memory Footprint	54.9MB	14.1MB
Modularity	Unified data, timing and routing control path	Distinct data, routing and handshake control path
Extendible	Difficult	Easy and efficient
Variability	Not modeled	Modeled

analyzing only the on-chip communication network, full system simulator can be overkill. Our work focuses on the on-chip communication network.

Noxim [6, 21] is a synchronous NoC simulator most suited for Wireless NoCs design. DARSIM [30] is a synchronous NoC simulator for input-queuing based routers providing speed-up relative to simulation of NoCs on a single core. BookSim2 [1, 27] is a synchronous NoC simulator for wired NoCs. BookSim2 gives a detailed description of the routers and the interconnect behavior for the NoC designers to test and validate their NoCs. BookSim2 allows design of the microarchitecture of the NoC routers and channels and reports the packet and network latency statistics. It can also be configured to report an estimate of power consumption by integrating Orion [44] power simulator.

PANE versus HNOCS

HNOCS [3, 14, 15] is a system-level synchronous/asynchronous NoC simulator built completely on the OMNeT++ framework. Synthetic traffic generation in HNOCS is done using in-built traffic generation functions in OMNET++. In HNOCS, unlike PANE, each module is responsible for carrying out its designated function as well as the asynchronous handshaking. Therefore, while in principle, HNOCS supports synchronous NoC simulation, it comes at the effort of re-implementation of the existing models in other simulators.

Unlike HNOCS, PANE uses the OMNeT++ framework only to implement its handshake-control path. PANE allows implementation of data and routing control (DAR) path in other synchronous NoC simulators. Decoupled modeling of handshake control from DAR paths in PANE makes the simulator modular. Thus, PANE can stay at par with its synchronous counterparts. While HNOCS benefits in terms of simulation speed by unifying the data, routing and handshake control paths, PANE presents a modular, hardware-like model of the systems.

In asynchronous systems, the time taken by each functional unit is expressed in real time unlike in synchronous systems where it is expressed as number of clock ticks. In addition to this configured delay, each functional unit also would experience data-dependent delays. PANE, unlike HNOCS, allows user to configure the NoC with delays for each functional unit as well as the variability due to data. Arbitration in asynchronous systems can also take an unequal amounts of time based on when the requests arrive. Unlike HNOCS, PANE models this behavior by detecting clashing requests and introducing arrival time dependent variable delays in releasing the grants for them. These features also allow modeling variabilities that arise in synchronous systems due to device variations and layouts.

In Table 1, we also show that HNOCS requires about 4x memory as compared to PANE and takes about the same time as PANE to simulate a 4×4 mesh NoC with uniform traffic. To compare the simulation time, we inject the same number of packets in the NoC and run the simulation until

Table 2. Summary of Comparison of Different System-level and NoC Simulators from the Literature

Simulator	Sim. Time (s)	Design space exploration	Synthetic Traffic	Benchmark Workloads	Async/Het NoC	Comparable to sync sim.
GEM5 [2, 17]	15	Limited	Yes	Yes	No	—
Multi2Sim [5, 41]	4	Limited	Yes	Yes	No	—
Sniper [10, 20]	—	Limited	No	Yes	No	—
BookSim [1, 27]	3	Extensive	Yes	Yes	No	—
Noxim [6, 21]	5	Extensive	Yes	Yes	No	—
HNOCS [3, 14, 15]	29	Extensive	Yes	No	Yes	No
PANE+BookSim2	**32**	**Extensive**	**Yes**	**Yes**	**Yes**	**Yes**
PANE	*6*	**Extensive**	**Yes**	**Yes**	**Yes**	**Yes**

the system drains these packets. We report the simulation times in synchronous mode for both HNOCS and PANE.

In Table 2, we summarize the comparison of different simulators. Simulation time is the time taken by the simulator to simulate an NoC. Although Gem5 and Multi2Sim are full-system simulators, for fairness, in this comparison we simulate them for NoC only. For PANE, we simulate the system for 100,000 epochs with each epoch 0.1ns long. The table shows the simulation time taken for simulation of an asynchronous system with the functional unit delays as 1, 2, 3, and 5ns each. When PANE simulates a synchronous NoC with each of the functional units taking 1ns to complete their operation, the system takes 34s to complete the simulation. Multi2Sim, Noxim and Booksim are configured to simulate a 4×4 mesh NoC for 100,000 cycles at 0.05 packets/router/cycle injection rate routing packets using dimension order XY routing. Design space exploration indicates the scope and ease of design space exploration supported by the simulator. Since Sniper does not support synthetic traffic generation for NoC simulation, we have not reported the simulation time for it in the table. As shown in the Async/Het column, only HNOCS and PANE support asynchronous and heterogeneous NoC simulation. However, in terms of functionality, unlike HNOCS, PANE is at par with the contemporary synchronous NoC simulators.

4 DESIGN OF PANE

PANE supports modeling, simulation, and analysis of synchronous (*sync*), asynchronous (*async*) and heterogeneous (*het*) NoCs. In PANE we have handles to separately model the data and routing control (DAR) path and handshake-control path in two different processes. In asynchronous paradigm, the DAR path is equivalent to datapath and handshake-control is equivalent to control-path in conventional synchronous systems. VC to be selected or the route to be taken are based on the routing-control path. Handshake-control path, however, is responsible for asynchronous handshakes, progress of data and event responsiveness. While PANE is capable of modeling its data path from scratch, to stay on par with the synchronous NoC simulators, PANE also allows augmenting other (modified) synchronous NoC simulators to model the DAR path. This decoupled architecture is more RTL-accurate and modular. Hence, unlike HNOCS [15], which was severely limited by the off-the-shelf functionality, PANE comes in with handles to integrate available NoC simulators.

For this discussion, we have used a modified BookSim2 [1, 27] to model the DAR path. However, PANE is a concept where we adapt the notion of a versatile plug-in that allows simulation of async, sync and het systems. This plug-in, though explained here in the context of NoCs and BookSim2, is flexible and has a well-defined interface to be used with other simulators as well. Additionally,

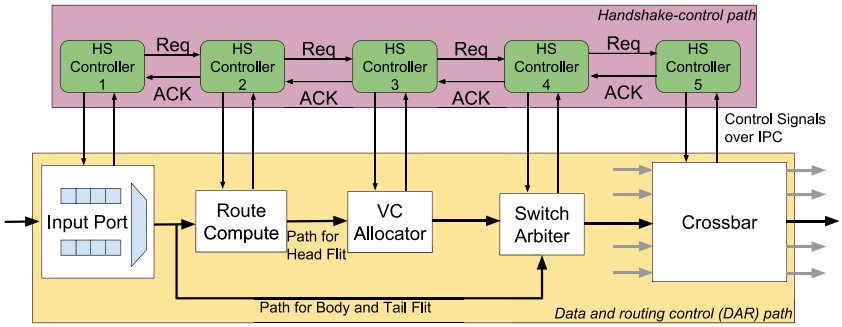


Fig. 2. Schematic of an asynchronous router as modeled in PANE.

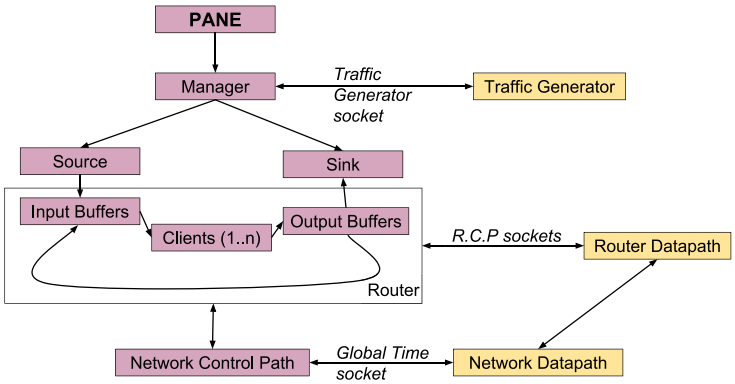


Fig. 3. Connection graph of modules in PANE simulator.

PANE also supports modeling of the functional units, thereby eliminating the IPC communication. This spectrum of configurations ranging from use of existing simulators to development of models from the scratch is provided by PANE.

Figure 2 shows the schematic of the asynchronous router in PANE, where the DAR paths are decoupled from the handshake control path. PANE's handshake-control units model the handshake controllers (green in Figure 2). When the handshake-control path and the DAR path are implemented by two different processes, the interaction between these is carried out over inter-process communication (IPC) sockets [26], as shown in Figure 2.

Figure 3 shows the interaction between different modules of PANE simulator. The DAR path modules (marked yellow in Figure 3), as described earlier, can be built in PANE or can be used from other synchronous NoC simulators. The DAR paths model generation and movement of data in the network. Hence, these modules can either generate real or synthetic traffic and control the packet injection into the network alongside describing the routing-control logic.

The modules in the handshake-control path are responsible for controlling the progress of flits through the NoC by passing "tokens." With every flit generated by the Traffic Generator (refer Figure 3) in the data path of the NoC, a corresponding *token* is generated by the PANE manager in the handshake-control path of the NoC. In other words, the movement of tokens in the handshake-control path governs the movement of flits through the data and routing control (DAR) path, as is implemented in hardware asynchronous pipelines.

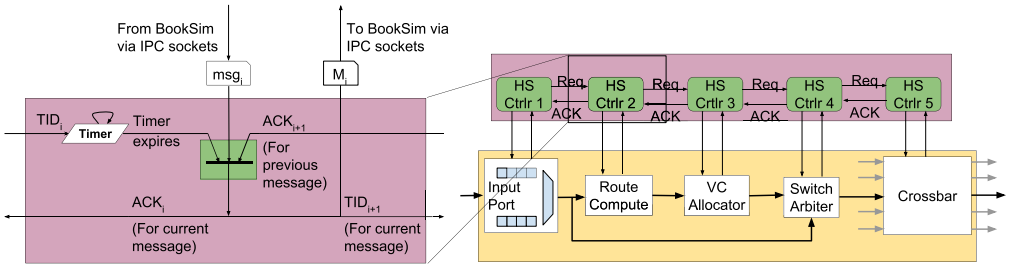


Fig. 4. Schematic of a handshake controller *(HS Ctrlr)* as modeled in PANE.

4.1 Modules in the Handshake Control Path of PANE

The handshake-control path of PANE consists of the following modules:

(i) Handshake-controller (HS Ctrlr): It is the smallest unit of asynchronous execution in a system. In the given description, each functional unit in the router is asynchronous to the other functional units. Input port, routing control unit, VC allocator, switch arbiter and crossbar are the functional units in a typical NoC router. Therefore, there is a handshake controller associated with each functional unit of of the router and is implemented as shown in Figure 4. The handshake-controller is composed of an asynchronous *JOIN* element that waits for the IPC message "msg" from the data path, an ACK from the downstream handshake-controller and for the timer to expire. The timer generates an event when the amount of time the functional unit is configured to take to perform its operation expires. When all the three events have occurred, the handshake controller generates an ACK to be sent to the upstream handshake controller and passes the token to the downstream handshake controller simultaneously, unlike synchronous systems where everything moves on the clock edge. When the token for the downstream functional unit is generated, the data path is notified over an IPC socket message about the progress of the token.

To allow modeling variability in PANE, the timer values can be configured accordingly. The handshake controller associated with the crossbar arbiter keeps a track of arrival time of the requests. Tokens are said to clash if they arrive too close in time to each other at the arbiter. Clash resolution is a user configurable parameter and is a property of the arbiter hardware implementation for a given technology. If the tokens clash at the arbiter, then the arbiter in hardware takes infinitely long time to resolve arbitration. In PANE, this "clash penalty" is also user configurable and is determined by the RTL description of the arbiter.

(ii) Token source: It releases new tokens into the system based on the IPC messages from the traffic generator in the data path. Traffic generator is the DAR module responsible for generating new traffic real or synthetic into the NoC based on the specification.

(iii) Token sink: From the destination, the flits retire into the sink to record message statistics. At the end of the simulation, each sink reports these statistics to the PANE manager for report generation.

(iv) PANE manager: PANE manager maintains PANE's notion of progress of time and synchronizes the operations between the data, routing-control and the handshake-control paths. It informs the respective *token src* of new token generation and collects statistics from the *token sink*.

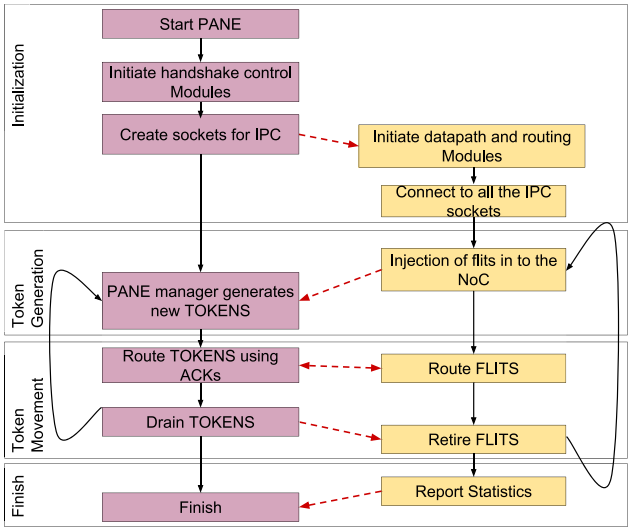


Fig. 5. Sequence chart for the operation of PANE. This chart shows sequence in which the modules in PANE are invoked.

The handshake-control path and the DAR path in PANE, when implemented on different platforms, interact at through inter-process communication (IPC) sockets. If the DAR path for PANE is implemented alongside the handshake-control path, then the IPC synchronization is eliminated at the cost of significant design effort. PANE models continuous evaluation of system events by evaluating the system at very short durations of time. In other words, the user defines the minimum time duration that they want to resolve in simulation. The system is evaluated every resolution time duration and this is called an *epoch*.

4.2 PANE Operation

Functioning of PANE can be described as the interaction between the DAR path and the handshake-control path. Figure 5 shows the interaction between the DAR path modules (shown in yellow) with the handshake-control modules (shown in pink). The solid line black arrows represent movement of information within the process while the dashed red arrows indicate IPC between the two processes. The interaction of the DAR modules with the handshake-control modules is as follows:

(i) ***Initialization:*** On initiating PANE (Figure 5), PANE manager creates $n = R.C.P + 2$ IPC sockets, where R is the total number of routers in the NoC, C is the number of handshake controllers per router and P represents ports per router. Routers in mesh and torus topology have 5 ports each. Two additional IPC sockets, as seen in Figure 3, are required for handling global time and traffic generation. This allows all the events at every port of each client in all the routers of the NoC to be kept a track of. PANE then spawns the DAR path process that then connects to the sockets created by the PANE manager. These IPC sockets are responsible for maintaining lock-step progress between the data and the handshake-control paths of the NoC.

PANE manager also sets up a system global timer to capture wall-clock time and configures the desired NoC based on the user inputs. The configurable NoC parameters include *physical attributes*, like the topology and buffer organization; *routing attributes*,

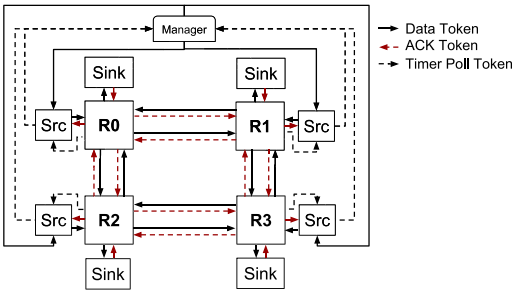


Fig. 6. Schematic of a 2×2 mesh network in PANE. The details of different types of messages are given in Section 4.3.

such as routing function and arbitration logic; *timing attributes*, such as functional unit delays, variability bounds, and arbiter clash resolution parameters; and simulation attributes, such as the workload specification and duration of the simulation. In case more that one clock domains are being modeled using PANE, unit and link delays for all clock domains need to be specified for appropriate set of routers. Once this initiation process is done, PANE is ready to simulate the configured NoC.

(ii) Token Generation: Based on the traffic pattern or workload to be simulated, injection rate set and the packet specification configured, the data path of PANE generates and injects flits into the NoC (Figure 5). During the simulation, all the flits that are injected into the network need to be tracked until they retire to avoid duplication, data loss or run-overs. This bookkeeping is also required for generating statistics at the end of the simulation. In a synchronous setting, this sequencing is easily handled with clocked operation modeled as iterative evaluation of the system. But in an asynchronous setting, the handshake controller (HS Ctrlr) needs to ensure sequencing and event-responsiveness. For every data flit generated in the DAR path, the handshake-control path generates a corresponding token. The information regarding the number of flits generated in the data path and their unique source IDs is passed to the handshake-control path through the appropriate IPC sockets. PANE-manager releases the tokens into the handshake-control path of the network through the corresponding *token src* nodes.

(iii) Token Movement: Token generated in the handshake-control path moves from the *token src* through multiple routers before it retires at the *token sink*. Movement of tokens in the handshake-control path is lock-stepped with the movement of flits in the data path (Figure 5). Flits are routed based on the routing control signals on the data path. However, the timing of the flit movement is controlled by handshakes on the handshake-control path through the corresponding tokens.

To simplify the understanding of token movement, let us take an example of a flit whose source is connected to data path of router R1 and destination is connected to R2 (Figure 6). The flit follows a two-router path. For this example, the PANE manager on the handshake-control path triggers the *token src* connected to *router[1]* that injects a token into *router[1]*. This token allows the corresponding flit to progress in R1. After the delay of R1, the token progresses to *router[0]* followed by the movement of the associated flit to R2. Route $router[1] \rightarrow router[0] \rightarrow router[2]$ is followed as the routing control is configured to route the flits using XY routing. Once the destination is reached, the flit retires and the token is consumed in the associated *token sink*. Thus, once a flit retires in the DAR path, its equivalent token is ejected from the handshake-control path as well.

Therefore the total number of tokens in the PANE handshake-control path at any point in time during the simulation is exactly the same as the total number of in-flight flits in its data path.

(iv) ***Finish:*** Generation, injection, routing and retirement of flits and tokens is done until the workload trace ends or the simulation duration are completed. Trace is a file containing a set of prerecorded NoC transactions. After the simulation is run to completion, the finish routine in the data path of the router reports the NoC statistics such as network latency, flit latency, throughput and so on. Since these statistics are obtained based on the actual module delays and not in terms of clock cycles, the asynchronous NoC performance can be measured. The datapath modules disconnect from the sockets and terminate following which the handshake-control path modules remove the sockets and exit.

4.3 Messages in PANE

Progress of tokens in the implementation on PANE is managed by passing messages between different modules. The messages in PANE control are as follows:

(i) ***Data/ACK token:*** Token messages are passed from one handshake-controller to another (or router) indicating passing of the tokens. Also, as the handshake-controller becomes available to accept new token, an ACK message is sent to its preceding handshake-controller indicating its availability to service new token. In PANE, the token messages move forward in the router while the ACKs flow in reverse. Latencies of data and acknowledgement is modeled as the delays of the token and ACK messages in PANE.

(ii) ***Timer poll token:*** Poll messages define the smallest amount of time that can be resolved by PANE, i.e. the duration of a single epoch. Handshake-controllers poll the PANE manager over a specified Input Port to synchronize the handshake-controller's notion of time. *Token src* polls the PANE manager to know if any new flits are being generated in the system by the data path. This information is used by the token src to release new tokens into PANE.

(iii) ***Socket messages:*** These messages are used to carry out the inter-process communication (IPC) between the handshake-control and the DAR path processes of PANE. BookSim2, which is used to model DAR path in this implementation of PANE, is modified to connect to sockets and to communicate with the handshake control path over these sockets.

4.4 PANE Router

The schematic of a 2×2 mesh NoC in PANE is shown in Figure 6. The routers communicate with each other via the TID-ACK handshake, where TID is the *Token ID*. The tokens in the router move in the handshake-control path while the flits in the data path. Movement of the tokens and flits along the NoC is in lock-step. The token and the flit are released into the system with the same ID. The token movement from *token src i* to *token sink j* follows the following path:

$$(PANE - manager) \rightarrow (src[i]) \rightarrow (router[i]) \rightarrow (router[p] \ldots router[q]) \rightarrow (router[j]) \rightarrow (sink[j])$$

Here the token goes through many routers starting from router[i] and ending at router[j], and flows through intermediate routers such as router[p] and router[q]. When the router[p] is empty of token, it sends an ACK signal (shown as red arrow in Figure 6) to router[i]. After the router delay time is elapsed, router[i] send the TID to router[p], if the flit has been processed by the data path. Thus in router, the TID is sent to the downstream router only when it has space to hold a new token.

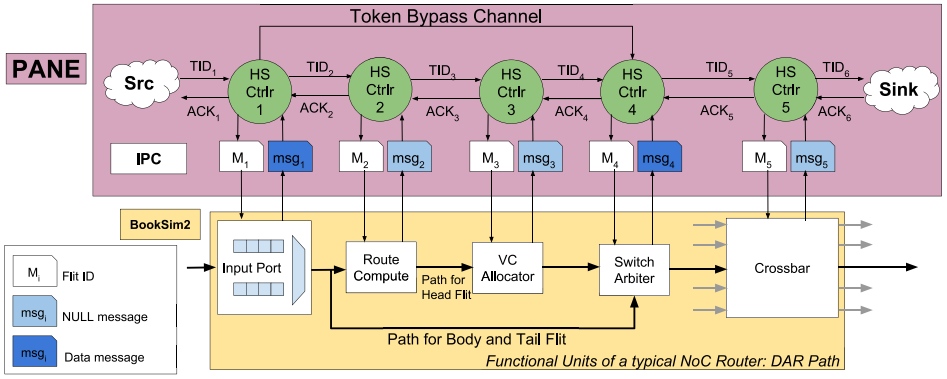


Fig. 7. Block diagram of a router modeled in PANE. The section of the router shaded yellow is the DAR path of the router while the one in pink is the handshake-control path. When implemented as two separate processes (default), these processes communicate over IPC sockets. The progress of flits in the DAR path is lock-stepped with the progress of associated token in the handshake control path. Token in the handshake-control path progresses from the handshake-controller (green boxes) as shown in Figure 4.

The functional units in the data path of a router are associated with handshake-controllers in the handshake-control path. Figure 7 shows the schematic of a router. The sequence of events followed by the handshake-controllers of a router are as follows:

(i) If handshake-controller1 is free, then a token enters handshake-controller1 from either the *token src* or from the handshake-controller5 of the previous router. After receiving the new token in handshake-controller1, the handshake-control path notifies the data path to allow processing the new flit entering the Input Buffer (shown as M1). After the data path has added the incoming flit to its buffer, it notifies the handshake-controller1 of the router about the ID and Type (Head, Body, Tail) of the flit that is to be read from the buffer (shown as *msg1*). If the buffer was empty when the flit entered, then *msg1* would contain the same id as *M1*.

(ii) The handshake-controller2 and handshake-controller3 are associated with the Route Compute and VC allocator units of the corresponding data path. Since in wormhole switching scheme, only the head flit is used to compute the route and the VC of a packet, the tokens corresponding to the head flits in data paths traverse through handshake-controller2 and handshake-controller3. The body and the tail flits use the bypass channel to contend for switch arbitration and allocation at handshake-controller4, as they would in hardware. Therefore, after the token stays in handshake-controller1 for its configured delay, depending of the type of flit being considered, the tokens from handshake-controller1 move to either handshake-controller2 or directly to handshake-controller4 using the bypass channel. In either case, the respective ACK signals are assessed for the availability of the handshake-controllers to accept new data. If the token is associated with a head flit (it moves to handshake-controller2), then the progress of flits to handshake-controller3 and then to handshake-controller4 is based of the completion of the handshake-controller's execution delay and the ACK from the next handshake-controller. Also *msg2* and *msg3* sent from handshake-controller2 and 3, respectively, are NULL messages.

(iii) At handshake-controller4, the tokens arbitrate for the output port in the crossbar. Handshake-controller4, in the current set-up, is associated with switch arbiter. As

asynchronous arbitration can take different amounts of time to arbitrate based on how close the requests arrive, the timer in the handshake-controller4 is designed to mimic this behavior in clashing requests. This is one of the key features of PANE.

(iv) Since handshake-controller5 requires an ACK from handshake-controller1 of the router associated with its respective output port in handshake-controller4, the data path informs handshake-control path of the output port for each flit it processes in *msg4*. Following wormhole switching scheme, after processing a token associated with the head flit, its corresponding body and tail flits (that would have arrived via the bypass channel) are processed. Tokens move from handshake-controller4 to handshake-controller5 after the data path completes its operation, the token has waited in handshake-controller4 for the programmed amount of time and the handshake-controller4 has received an ACK from handshake-controller5.

(v) From handshake-controller5, depending on the output port of the token, the token moves to handshake-controller1 of another router or to the token sink. If the token moves to the handshake-controller1 of another router, then steps 1–3 are repeated until the token reaches the sink.

BookSim2 is modified such that the creation, progress and retirement of flits generate socket messages in sockets associated with the respective port at the associated functional unit in the respective router. Modules dealing with progress of flits in BookSim2 are also modified to wait for appropriate socket messages from the handshake control path of PANE.

PANE is an implementation of the concept of extending existing simulators to support asynchronous system simulation. In this work, though explained in the context of a NoC simulator BookSim2, the attempt has been to highlight the structure, operation and interfaces of PANE. This is because PANE is a flexible and versatile simulator that allows designers to borrow existing designs from other simulators without the re-design effort. The IPC mechanism, despite its simulation overheads, is a mechanism that can allow integration of other simulators also to support asynchronous designs. Having a plug-in with a well defined interface description potentially allows extension of any existing simulator to the asynchronous design domain. Since this mechanism of providing an enhancement package instead of a full-fledged simulator is not explored for asynchronous simulators earlier, PANE tries to experiment with this approach.

4.5 Notion of Time in PANE

The time taken by functional units in synchronous setting is measured in clock cycles. However, asynchronous circuits do not have a notion of clock cycles. The handshakes, functional unit delays, arrival time dependent arbitration delays and variability mechanism are modeled in PANE. After the event has been registered, the handshake-control path of PANE invokes the appropriate functional unit to perform the operation. To maintain the lockstep progress of handshake-control path with that of the DAR path, the two need to synchronize periodically. The period of this synchronization is called an *epoch* that is the minimum time that can be resolved between two events. This duration is user-configurable and is ideally much smaller than the evaluation time of the functional units in the system. However, smaller epoch duration implies a finer resolution of events in the system leading to a longer simulation time. If the epoch is made equal to the clock period of a synchronous system, then the simulation time of PANE can be significantly reduced. Every epoch, the PANE manager polls the hs-ctrlrs to check if there are any actions to be taken. Thus, polling mechanism helps in implementing the notion of global real time, as opposed to the notion of a number of cycles implemented in synchronous NoC simulators.

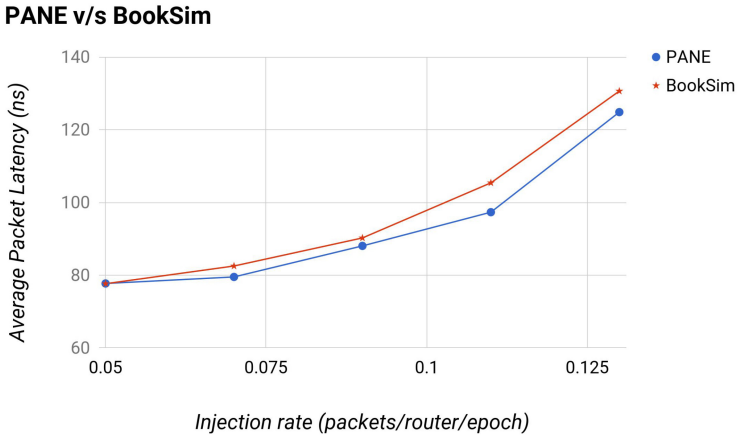


Fig. 8. Validating PANE: Average packet latency for PANE and BookSim2 8×8 mesh networks for Uniform traffic pattern at different injection rates. BookSim2 is a synchronous mesh NoC simulator that has been validated against RTL by the authors of BookSim2. PANE is configured to simulate synchronous NoC for this comparison. The difference in the reported average packet latency by BookSim2 and PANE is ~2%.

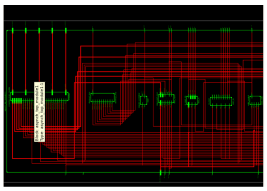
The epoch delays come from the router design and layout. PANE also allows wire delays can be augmented to the unit delays or be specified as channel delays separately. In principle, PANE can support modeling of functional unit delays as a combination of gate delays. However, every message on each of these connections generates an event in PANE. Therefore, modeling of functional units at gate level will take very long, which can be undesirable in a system-level study of NoCs. However, if required, gate-level models can also be used. Interestingly, PANE architecture also allows some of the blocks to be modeled at gate-level and the others at block level. This can be helpful when looking at the design impact of one block on the overall system performance without paying additional simulation time penalty for the other functional blocks in the system.

5 VALIDATING PANE WITH BOOKSIM2

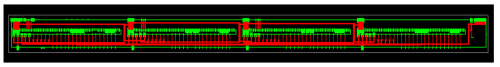
In this section we validate PANE for correctness and correlation with respect to the RTL. Since PANE can be used to simulate synchronous as well as asynchronous NoCs, the validation is also carried out for synchronous as well as asynchronous modes of operation. For synchronous mode, we validate PANE against an existing synchronous NoC simulator for correctness. BookSim2 [1, 27] has been validated against RTL and hence can be used as a reliable reference for validation. The PANE implementation, for the validation experiment, is such that each PANE handshake-controller takes unit time to complete its action. In other words, PANE is configured to simulate a synchronous NoC. Since this is the same as simulating a NoC in BookSim2, the results obtained after both the simulations should match. This is shown in Figure 8, where, for Uniform traffic pattern at different injection rates, the average packet latencies for both the simulators match until saturation is reached.

The validation of PANE with BookSim2 is carried out for an 8×8 mesh for uniform traffic routed using dimension order XY routing. The routers for this validation have four virtual channels on each port. Each virtual channel can hold up to four flits. The routers are configured for wormhole switching. Each packet in the network is one flit long.

For validation of PANE in asynchronous mode, we have implemented and synthesized the asynchronous NOC in UMC 65nm. Figure 9(a) shows the RTL schematic of a router implemented in UMC 65nm. An asynchronous 2×2 NoC is designed using this router and the handshake



(a) RTL of a single asynchronous router with 5 I/O ports, 4 VCs of depth 4 each.



(b) RTL of the 2 × 2 asynchronous NoC.

Fig. 9. RTL for the asynchronous router and NoC developed in-house implemented in UMC 65nm technology.

Table 3. Comparison of Packet Latency in RTL and PANE Simulations

	One-hop latency	Two-hop latency	Three-hop latency
Async RTL	1.48	2.92	4.36
PANE	1.48	2.96	4.44

controllers. The RTL schematic of the same is shown in Figure 9(b). We have obtained the average packet latency using post-synthesis simulations. The delays obtained from RTL are used in PANE to obtain the simulation traces for a 2 × 2 NoC. These traces are then used to generate a testbench that is used for RTL simulations. The average packet latency for a packet that hops through single router, two routers and three routers are shown in Table 3. The packet latency reported for three-router path is 4.44ns in RTL simulations, and 4.36ns in PANE simulations. Thus, the error between PANE and RTL simulations is less than 2%. In our router module implementation in RTL, there is a constant 0.04ns delay that gets added for inter-router communication, which is absent in PANE. The 2 × 2 NoC, when implemented in the RTL using the UMC 65nm technology, occupies a total cell area of 937.66μm^2.

5.1 Verification of PANE

PANE is designed to model and simulate handshake-based communication between the functional units of the NoC in an event-driven fashion. Also, in PANE, we have two sets of paths: one for the movement of the data and the credits and the other path for the movement of tokens and the ACKs. The former path, in this article, is referred to as the DAR path and the latter is called the handshake-control path. As both of these paths have a forward channel (the data channel for the DAR path and the token channel for the handshake control path) as well as a backward channel (the credit channel for the DAR path and the ACK channel for the handshake control path), in this section we evaluate PANE for freedom from deadlocks, livelocks and correctness. Communication between the DAR path and the handshake control path is carried out by passing (socket) messages.

A system is considered deadlocked when the system cannot progress its state due to a circular dependency between resources being requested for. In case of the DAR path, the system changes its state only when either a data flit progresses. Since in the construction of PANE, the mechanism of movement of data flit is not modified, if the DAR path is deadlock-free in isolation, it remains deadlock free when augmented with PANE. The handshake control path of PANE is responsible to notify the DAR path about when to progress the data flit. The handshake-control path changes it's state on movement of tokens and ACKs. The tokens progress from their source to their sink when the handshake-controller receives an ACK from the receiving handshake-controller. This ensures

Table 4. Scope of Architectural Analysis Carried Out in This Paper Using PANE

(a) Default system parameters for evaluation

Parameter	Value
NoC Size	4×4
NoC Topology	Mesh, Torus
Routing Algorithm	DOR XY
Traffic Patterns	Uniform Traffic
Injection Rate (Network load)	0.05 pkts/router/ns or (pkts/router/cycle)
Packet Size	1 flit
Number of VCs	4 flits
VC Depth	4 flits
NoC Warm-Up period	30,000 epochs
Simulation period	100,000 epochs
Switching Scheme	Wormhole

(b) Summary of experiments performed

Experiment	Sweep Range	Fig. Number
Synthetic Traffic	Neighbor etc.	10
Real Benchmark	PARSEC [16]	11
Topologies	Mesh, torus	10, 11
Network Sizes	2×2 to 32×32	12
Simulation Time	2×2 to 16×16	13
Buffering	1×32 to 32×1	14
Routing	DOR etc.	15
Injection Rates (Network load)	0.05-0.95	16
Packet Sizes	1-5 flits	17
Epochs	1-10	18
Power	Neighbor etc.	19
Router Architectures	Neighbor etc.	20

This analysis aims to motivate the use of PANE for system-level analysis like the synchronous NoC simulators.

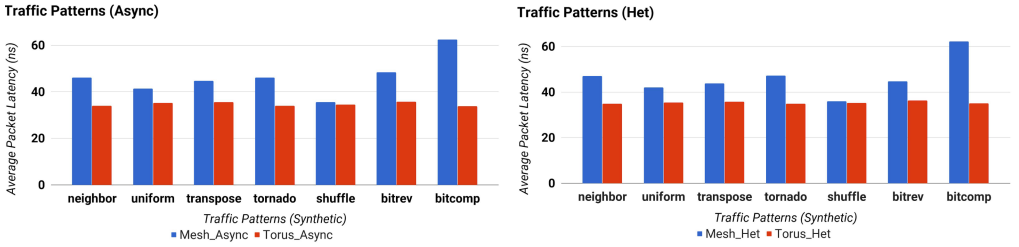
that no tokens are run-over in the system. Also, the handshake controller waits for a message from the DAR path to allow the progress of the token. This ensures that the token moves only when it has actually completed its specified task in the functional unit. In other words, the token is permitted to progress only when the flit is ready to progress. This ensures that no token runs-ahead of its respective data flit in PANE. When the handshake-controller allows the token to progress, like in real hardware, it also ACKs the preceding handshake controllers. This indicates the availability of the newly freed handshake controller to process new incoming token. The handshake controller also signals the DAR path to progress the flit and update its state to stay in lock-step with the token. Since the token is allowed to move only where the flit moves, and the movement of flits in the DAR path is assumed to be deadlock free, the movement of the tokens is also deadlock free. ACKs move from the sink to the source in the handshake-control path in PANE. ACKs shadow the credits as tokens shadow the flits. Thus, like the tokens, the ACKs are also deadlock, run-over and run-ahead free. Similarly, if the DAR path for the NoC modeled in PANE is independent of livelocks, the handshake-control path of PANE is also livelock-free.

These properties were also verified for PANE in **NuSMV** models. We modeled the handshake controller as a module that generates a token and an ACK when it receives a token and an ACK. These handshake controller modules are connected to form a 2×2 NoC such that the token moves forward (source to sink) and the ACK moves backward (sink to source) in the system. Each of these handshake-controllers can operate asynchronously and hence to to model the event-driven behavior, the handshake-controller modules are instantiated as processes. This setup is then queried for the possibility of generation of the token or the ACK to be forwarded before the arrival of the token **and** the ACK in the handshake controller using the linear temporal logic.

6 PANE USE CASES AND ANALYSIS

In this section, we provide use cases and their architectural analyses for PANE to compare sync, async and mixed sync-async (het) NoCs for system-level parameters, which was not possible earlier.

The motivation of this section is not to compare sync versus async systems but to show the utility of PANE. The default system configuration parameters are given in Table 4a. For the sake



(a) Async NoC

(b) Het (Mixed sync-async) NoC

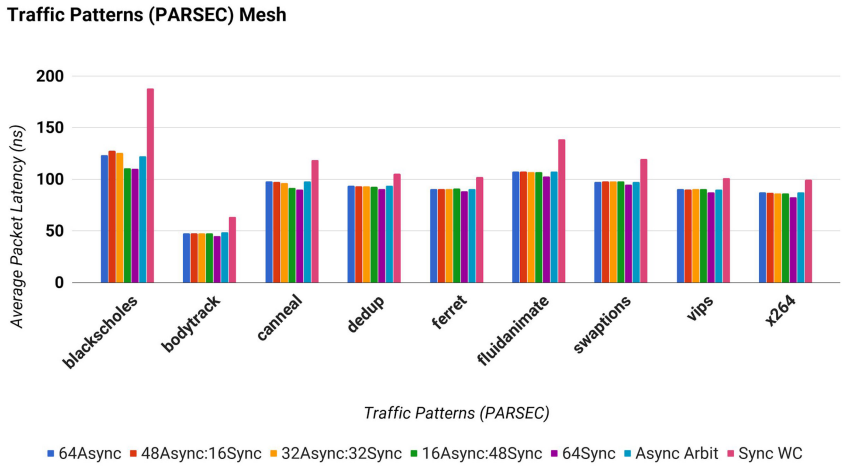
Fig. 10. Plot of average packet latency across 4×4 mesh and torus NoC for synthetic traffic of one-flit-long packets injected at 0.05 packets/router/epoch simulated in PANE for 100,000 after a NoC warmup for 30,000 epochs using DOR routing.

of experimentation, we consider an async router with the following latencies: input buffering stage is 1ns, the RC is 2ns, VC is 3ns and switch arbitration and crossbar is 5ns. For the sync routers, for fairness, we consider each stage to take 3ns to compute. To demonstrate the timing in realistic synchronous systems where the functional unit delays are same as the functional unit delays in the chosen async configuration, we also consider a worst-case sync (Sync WC) configuration. In the Sync WC configuration, clock duration is determined by the delay of the slowest pipeline stage in the router. Thus, each pipeline stage in the Sync WC router takes 5ns to compute its output. To also demonstrate the scenario where each stage can be affected by variability, in the Sync+ and Sync++ configuration, we configure each pipeline stage in the router to have 5% and 10% variability. We also model a heterogeneous square (Het) NoC where the routers in the upper half of the NoC are asynchronous and the ones in the lower half are synchronous. Unless specified otherwise, configurations from Table 4a are used. Table 4b summarizes all the use cases described below.

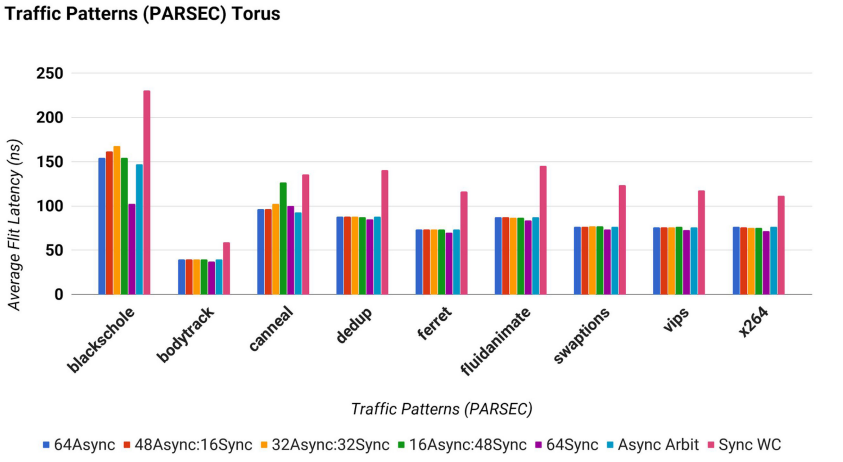
6.1 Analysis of NoC Parameters

In this section, PANE is used to analyse NoC with varying NoC parameters.

- **(i) Topologies:** In Figure 10(a) and (b), we use PANE to simulate mesh and torus NoCs. As torus topology adds extra links between the edge routers to the mesh, the mean path length between the nodes reduces. However, depending on the pair of communicating nodes, this change of mean path length may not reflect in the average flit latency. In case of bitcomp traffic pattern, we see that additional links lead to a significant performance advantage when connected as torus over as a mesh. However, this difference is negligible in shuffle traffic. This is because the path followed by most of the flits in torus as well as mesh is identical, giving little advantage to the additional links in torus.
- **(ii) Synthetic and Real Benchmarks:** PANE supports synthetic traffic and trace-based workloads. Figure 10(a) and (b) show the average packet latencies for different synthetic traffic patterns while Figure 11(a) and (b) for 8×8 mesh and torus NoCs for applications from the PARSEC [16] suite. Sync WC router has all its functional units clocked at the worst-case delay (5ns). Async arbit configuration uses routers that have a clash resolution of 5% and a penalty of 15–25 epochs. We observe that when considering the worst-case delay, sync NoC is the most under-performing configuration. We also observe that despite a fairly large clash penalty, the overall impact is not deterrent. This is because, in most real applications, the clashes do not happen as often.
- **(iii) Network sizes:** In Figure 12(a) and (b), PANE is used to simulate for networks of different sizes. In Figure 12(a) and (b), we plot the average packet latency in the NoC for one

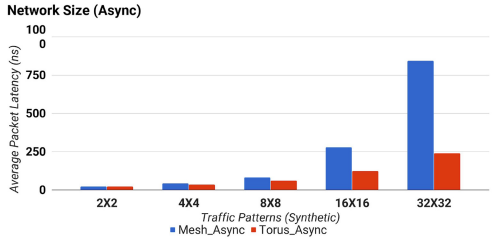


(a) Mesh NoC

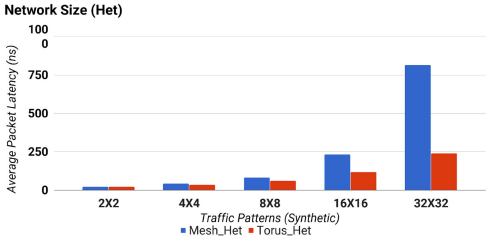


(b) Torus NoC

Fig. 11. Plot of average packet latency across 8 × 8 Sync, Async, and Het (mixed sync-async) NoC for PARSEC traffic using DOR routing.



(a) Async NoC



(b) Het (mixed sync-async) NoC

Fig. 12. Plot of average packet latency NoC for uniform traffic using DOR routing of different network sizes in mesh and torus topologies for one-flit-long packets injected at 0.05 packets/router/epoch simulated in PANE for 100,000 after a NoC warmup for 30,000 epochs.

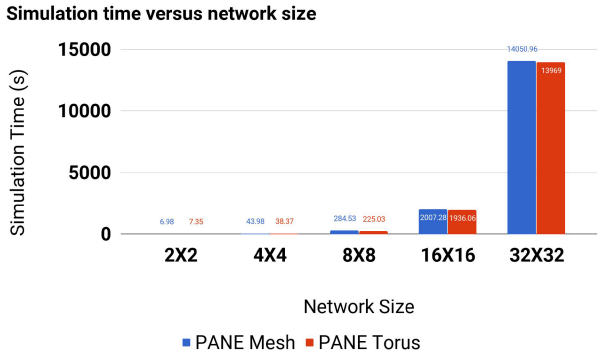
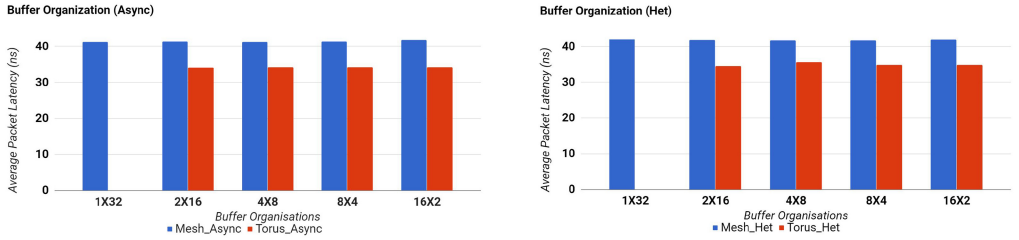


Fig. 13. Plot comparing the simulation time (s) for mesh and torus NoCs of different sizes from 2×2 to 32×32 for uniform traffic of one-flit-long packets injected at 0.05 packets/router/epoch simulated in PANE for 100,000 after a NoC warmup for 30,000 epochs using DOR routing.



(a) Async NoC

(b) Het (mixed sync-async) NoC

Fig. 14. Plot of average packet latency across 4×4 mesh and torus NoC for different buffer organizations ranging from 1×32 to 16×2 for uniform traffic injected at 0.05 packets/router/epoch where each packet is one-flit long and is router using the DOR routing. PANE is simulated for 100,000 after a NoC warmup for 30,000 epochs.

flit packet following uniform traffic for NoCs of different sizes in Async and Het NoCs, respectively. We see that packets in the mesh topology experience higher latencies as compared to the packets in the torus connected networks. This is to be expected, because torus has an additional wraparound link. In Figure 13, we plot the scaling of simulation time with an increase in network size. We see that simulation time increases drastically as the number of nodes increase in the system for both mesh and torus topologies. This increase is due to a large number of IPC messages generated in the system as described in Section 3.2 and also because a larger network has more events to be tracked. As PANE is an event-sensitive simulator, the performance of PANE takes a hit when the number of events to be tracked increases. Simulation speed can be increased by increasing the epoch duration.

(iv) Buffering organizations: In Figure 14(a) and (b), PANE is used to simulate for a different number of VCs and depth of virtual channels on the average packet latency. Since the resolution of deadlocks in torus topology requires at least 2 VCs, we do not plot the average packet latency for the router with 1 VC in torus topology. Deeper buffers at the input port accommodates larger packets while a large number of VCs allow more active flows. The plots in Figure 14(a) and (b) show the average packet latency for mesh and torus topologies for async and het routers for packets of size 1. As this packet can

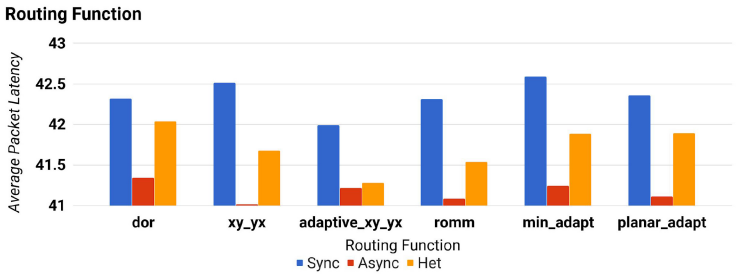


Fig. 15. Plot of average packet latency across 4 × 4 mesh for Sync, Async, and Het (mixed sync-async) NoC for uniform traffic injected at 0.05 packets/router/epoch where each packet is one-flit long using different routing algorithms. PANE is simulated for 100,000 epochs after a NoC warmup for 30,000 epochs.

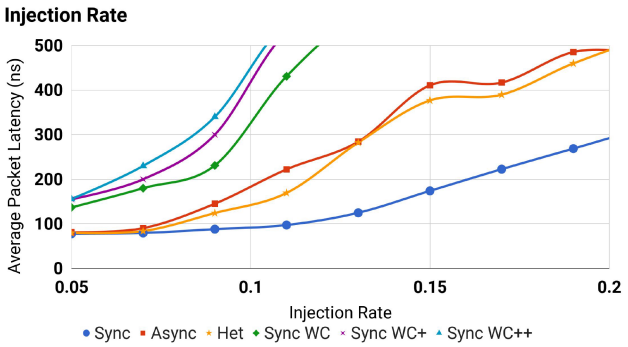


Fig. 16. Plot of average packet latency across 8 × 8 mesh for Sync, Sync WC, Sync WC+, Sync WC++, Async, and Het (mixed sync-async) NoC for uniform traffic using DOR routing at different injection rates. The packets are one-flit long and PANE is simulated for 100,000 epochs after a NoC warmup for 30,000 epochs.

always be accommodated in the buffering organizations being considered, we observe that async and het topologies show identical latencies at low load.

(v) Routing algorithms: In Figure 15, PANE is used to simulate for different routing algorithms. We observe that async routers outperform het and sync routers irrespective of the routing algorithm being used. This is despite the two routers configured to take an equal amount of time to process the flits. We also observe that async configuration in an adaptive routing setting has a greater advantage. Though nominal, this difference can be attributed to the elastic nature of async pipelines. Sync WC configuration, if plotted, is the most under-performing one, as expected.

(vi) Injection Rates: In Figure 16, PANE is used to simulate mesh and torus networks for different traffic patterns. In the sync configuration, the average packet latency rises steadily as the network load increases. However, in async and het configurations, the increase in average packet latency with increasing load is steeper. One of the primary contributors to this effect is the data dependent and arrival-time dependent arbitration delays. With more packets in the network, there is a greater possibility that the packets clash at the switch arbiter. Arrival-time dependent arbitration delays are discussed later in this section. We also plot sync configuration where we model variability. As discussed earlier in this section, in Sync+ and Sync++, we introduce a variability of 5% and 10% to the Sync WC functional unit delays respectively. We see that the performance further degrades and the network saturates at an even lesser load. PANE enables analysis of

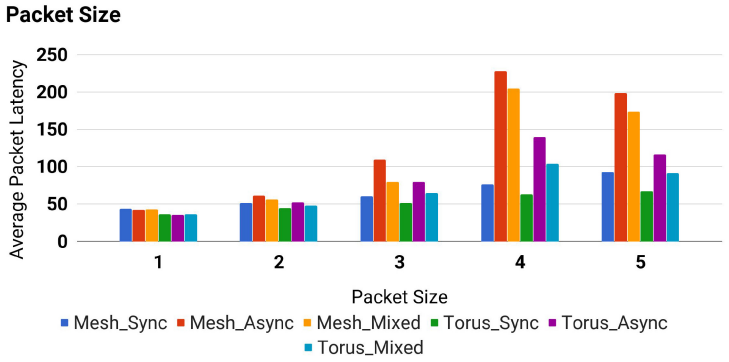


Fig. 17. Plot of average packet latency across 4×4 mesh for Sync, Async, and Het (mixed sync-async) NoC for uniform traffic injected at 0.05 packets/router/epoch using with traffic of different packet sizes. The traffic is routed using DOR routing and PANE is simulated for 100,000 epochs after a NoC warmup for 30,000 epochs.

Table 5. Effect of Arbitration Dependent Delays on Packets of Different Sizes

(a) Effect of clash penalty

	Penalty	9–11	45–55	90–110
Packet Size 1	APL	42.27	42.34	42.86
	MPL	107	135	185
Packet Size 5	APL	195.93	200.87	205.75
	MPL	681	729	888

(b) Effect of clash resolution

	Clash resolution	1%	5%	10%
Packet Size 1	APL	42.27	42.34	42.38
	MPL	107	107	107
Packet Size 5	APL	195.9	197.79	200.08
	MPL	640	771	808

Packet latencies are measured in ns. We report the average packet latency (APL), and the maximum packet latency (MPL) for each configuration.

system impact of variability, which is becoming a concern, especially in lower technology nodes. As PANE captures the wall-clock execution time, PANE can also be used for near threshold voltage (NTV) design at system and architectural level.

(vii) Packet Sizes: PANE supports packets of different sizes (Figure 17). The VC depth is set to hold a max of five packets for this experiment. As observed in Figure 17, for small packets, sync, async and het configurations of the NoC perform nearly as good. However, as the packet size increases, despite sufficient buffering at the input port, the average packet latency for async (and hence for het) NoC configurations is much larger. This is because, even at the same injection rate (which is measured in packets/router/epoch), there is a larger number of flits being generated in the network. These flits contend for arbitration and on a clash, can see large additional delays. Arbitration delays are discussed later in this section.

6.2 Analysis of PANE Features

In this section, different features of PANE are used to asses the NoCs.

(i) Arbitration Dependent Delays: Asynchronous arbiters are pushed to metastable state when the incoming requests clash. In real circuits, the amount of time that the arbiter

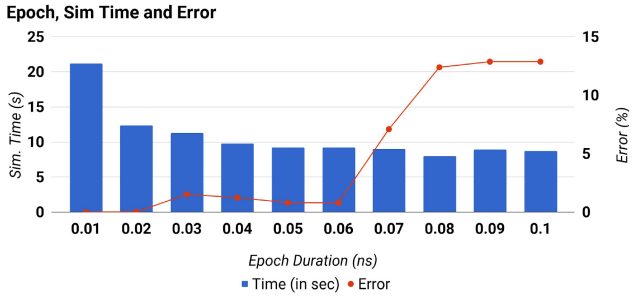


Fig. 18. Plot of simulation time (bar graph) and error (line graph) for different epoch durations for a 4×4 mesh for Sync NoC for uniform traffic using with traffic of one-flit-long packets. The traffic is routed using DOR routing and PANE is simulated for 100,000 epochs after a NoC warmup for 30,000 epochs.

considers as "too close" as well as the amount of time it can potentially take to resolve the clash is a property of the circuit. Thus, to model the arbitration dependent delays, in PANE we have introduced a parameter that allows the user to set the "closeness" of the arrival of two requests to consider them as a clash and a parameter to allow the user to set the penalty in case the arbiter sees a clash. The "closeness" parameter is described as the percentage of the epoch and is by default set to 1%. Thus, if the epoch is set as 1ns and if two requests arrive at the arbiter less than 0.01ns apart from each other, the arbiter considers them as a clash. When such a clash occurs, the time penalty is added. The penalty is measured in terms of the number of epochs and can be expressed as a range between $penalty_{min}$ and $penalty_{max}$ or a function, which is user configured. In Table 5, we report the average and maximum packet latencies of packets made of one flit and five flits for different clash resolution and penalty configurations. As is clear from the data, the effect of arbitration dependent delays is more profound in larger packets.

(ii) **Epoch Duration:** PANE is an event-driven simulator that simulates the events occurring in the network as they occur. With the DAR and handshake-control paths modeled in two different processes, responding to every event in the NoC as it occurs generates a large number of IPC. IPC is one of the slowest components of the simulator. Therefore, one of the ways to speed up the simulation is to increase the duration of epoch. Longer epoch duration implies that the events are not tracked as soon as they occur but are resolved at every epoch. This leads to faster simulations due to fewer IPC messages at the cost of loss of accuracy. This is shown for a 4×4 mesh in Sync NoC for uniform traffic in Figure 18. It is clear from the plot that the error is under 5% in simulations carried out with epoch durations 0.06 that takes about 10s to complete. BookSim2 on the other hand takes 3s to simulate a synchronous 4×4 mesh NoC for uniform traffic at 0.05 packets/router/cycle to simulate using DOR routing.

(iii) **Power:** PANE, like the contemporary synchronous NoC, supports power analysis of the modeled NoCs. As opposed to synchronous NoC where the power is consumed every clock tick, in asynchronous NoCs, power is consumed only when there is an activity on the functional unit or link. In systems like NoCs, asynchronous behavior can lead to significant power savings as shown in Figure 19. Orion, which is already integrated with BookSim, is used for power calculations in PANE. For asynchronous NoC, the power calculations are carried out based on the activity factor of the functional units in the NoC.

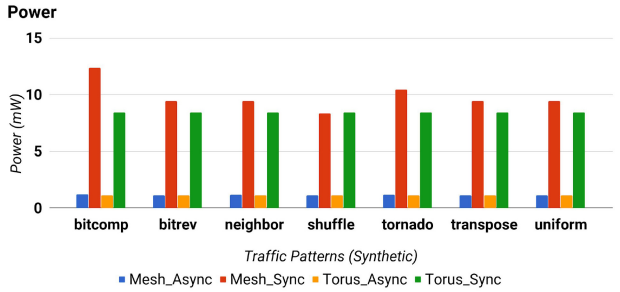


Fig. 19. Plot of power for different sync and async configurations of 4×4 mesh and torus NoC for uniform traffic using with traffic of one-flit-long packets. The traffic is injected at 0.05 packets/router/epoch and the NoC is simulated in PANE for 100,000 epochs after a NoC warmup for 30,000 epochs.

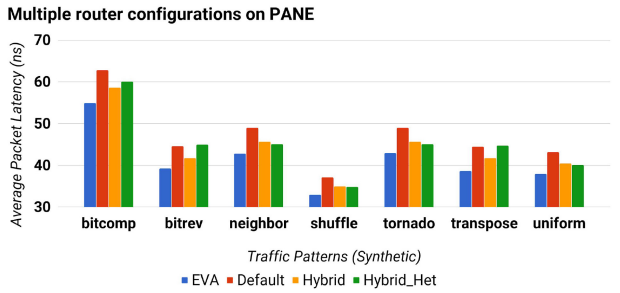


Fig. 20. Plot of average packet latency for EVA, default and hybrid routers in 4×4 mesh for synthetic traffic patterns using with traffic of one-flit-long packets.

(iv) Router architectures: This article discusses the default off-the-shelf configuration of the NoC router in PANE. However, PANE is not restricted to this model of router and can model a wide range of routers. As an example, a 4×4 mesh NoC built using dynamic VC allocating router [43] has been modeled in PANE. Figure 20 we compare the average packet latency in an NoC made only of EVA routers, made only of the default routers, a hybrid NoC and hybrid het NoC. In the hybrid NoC the top half of the NoC is made of the EVA routers and the bottom half using the default routers. In the hybrid het configuration the EVA routers are async and the default routers are sync. It is possible in PANE to model and simulate NoCs with more than one kind of routers in PANE. The VC allocator in the default router is configured to take 2 epochs to compute.

In summary, we have demonstrated the spectrum of work that can be done by PANE that was never reported earlier for NoCs. While PANE supports simulation of asynchronous networks, it can be optimized for speed. We observe that the simulation time for PANE is about 10 times that for a synchronous NoC simulation in BookSim2, as shown in Table 2. This increase in time is mainly due to the IPC communication and is a point of focus for the next version of PANE.

7 CONCLUSION

Pluggable Asynchronous Network on chip (PANE) simulator is the first tool that allows simulation of synchronous, asynchronous and mixed sync-async NoCs. This tool finds its application in cases where some design decision made for asynchronous NoC needs to be evaluated for its system-level impact for performance and power. So far, while this has been possible for synchronous NoC,

asynchronous and heterogeneous NoCs have been compared by their RTL area, power, and delay. We have presented a case where RTL comparison of routers solely based on their RTLs is not sufficient, and a system-level impact of router designs are necessary to make better design decisions. PANE has also been validated against BookSim2. We also demonstrate the use cases for PANE in this article. As PANE is a simulator, the use-cases provided in this article are only indicative of functionality that can be achieved using the PANE framework. Modular, scalable and portable design of PANE provides asynchronous NoC and SoC designers a platform to prototype NoC schemes allowing a sweep of design space. Being based on the idea of exclusion of the data-path from the handshake-control path, PANE supports better debugging, easier prototype development and more hardware-like modeling of NoCs for asynchronous systems. Design of PANE also makes it suitable to model systems for variability. We observe that the simulation time for PANE is about 3× that for a synchronous NoC simulation in BookSim2 with error under 5%. This is due to a large amount of IPC between the DAR path and the handshake control path. Reducing the number of IPC socket messages is a key point in the future version on PANE. Also, the error between PANE and the RTL simulations for asynchronous mode is found to be 2%.

ACKNOWLEDGMENTS

The work is supported by Intel India PhD fellowship, and partially supported by grant received from Ministry of Electronics and Information Technology (MEITY), Government of India for Special Manpower Development Project for Chips to System Design (SMDP-C2SD) and Research work undertaken in the project under the Visvesvaraya PhD Scheme of MEITY, Government of India.

REFERENCES

[1] [n.d.]. BookSim2 Simulator. Retrieved from http://nocs.stanford.edu/booksim.html.
[2] [n.d.]. Gem5 Simulator. Retrieved from gem5.org.
[3] [n.d.]. HNOCS Simulator. Retrieved from http://hnocs.eew.technion.ac.il/.
[4] [n.d.]. IVerilog. Retrieved from http://iverilog.icarus.com/.
[5] [n.d.]. Multi2Sim Simulator. Retrieved from http://www.multi2sim.org/.
[6] [n.d.]. Noxim Simulator. Retrieved from http://noxim.sourceforge.net.
[7] [n.d.]. Nvidia Tegra 2. Retrieved from http://www.nvidia.com/object/tegra-superchip.html.
[8] [n.d.]. Nvidia Xavier. Retrieved from https://blogs.nvidia.com/blog/2016/09/28/xavier/.
[9] [n.d.]. OMNeT++. Retrieved from https://omnetpp.org/.
[10] [n.d.]. Sniper Simulator. Retrieved from http://snipersim.org/w/The_Sniper_Multi-Core_Simulator.
[11] 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 *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS'09)*.
[12] A. Bardsley and D. A. Edwards. 1999. *Balsa: An Asynchronous Circuit Synthesis System*. Technical Report.
[13] Peter A. Beerel, Recep O. Ozdag, and Marcos Ferretti. 2010. *A Designer's Guide to Asynchronous VLSI*. Cambridge University Press.
[14] Yaniv Ben-Itzhak, Eitan Zahavi, Israel Cidon, and Avinoam Kolodny. 2011. NoCs simulation framework for OMNeT++. In *Proceedings of the 5th ACM/IEEE International Symposium on Networks-on-Chip (NOCS'11)*.
[15] Yaniv Ben-Itzhak, Eitan Zahavi, Israel Cidon, and Avinoam Kolodny. 2012. HNOCS: Modular open-source simulator for heterogeneous NoCs. In *Proceedings of the International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (ICSAMOS'12)*.
[16] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. The PARSEC benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International Conference on Parallel Architectures and Compilation Techniques (PACT'08)*.
[17] Nathan Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, Joel Hestness, Derek R. Hower, Tushar Krishna, Somayeh Sardashti, Rathijit Sen, Korey Sewell, Muhammad Shoaib, Nilay Vaish, Mark D. Hill, and David A. Wood. 2011. The Gem5 simulator. *SIGARCH Comput. Arch. News* 39, 2 (Aug. 2011).
[18] T. Bjerregaard. 2006. Implementation of guaranteed services in the MANGO clockless network-on-chip. *IEE Proc. Comput. Dig. Techn.* 153 (Jul. 2006). Issue 4.

[19] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. 2004. QNoC: QoS architecture and design process for network on chip. *J. Syst. Arch.* 50, 2–3 (Feb. 2004).
[20] Trevor E. Carlson, Wim Heirman, and Lieven Eeckhout. 2011. Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulation. In *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis (SC'11)*.
[21] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. 2018. Improving energy efficiency in wireless network-on-chip architectures. *ACM J. Emerg. Technol. Computing Syst.* 14, 1 (2018), 1–23.
[22] Rostislav (Reuven) Dobkin, Ran Ginosar, and Avinoam Kolodny. 2009. QNoC asynchronous router. *VLSI J.* 42, 2 (Feb. 2009).
[23] Amit Dua, Adam Sherer, and Umer Yousafzai. [n.d.]. Hardware Simulator Performance Scaling to Meet Advanced Node SoC Verification Requirements.
[24] Chris Fallin, Chris Craik, and Onur Mutlu. 2011. CHIPPER: A low-complexity bufferless deflection router. In *Proceedings of the 17th International Symposium on High Performance Computer Architecture (HPCA'11)*.
[25] Marisabel Guevara, Benjamin Lubin, and Benjamin C. Lee. 2014. Market mechanisms for managing datacenters with heterogeneous microarchitectures. *ACM Trans. Comput. Syst.* 32, Article 3 (2014), 31 pages.
[26] Brian "Beej" Hall. [n.d.]. Beej's Guide to Unix IPC. http://beej.us/guide/bgipc/output/html/multipage/unixsock.html. December 1, 2015.
[27] Nan Jiang, Daniel U. Becker, George Michelogiannakis, James D. Balfour, Brian Towles, David E. Shaw, John Kim, and William J. Dally. 2013. A detailed and flexible cycle-accurate network-on-chip simulator. In *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS'13)*.
[28] Ken Kundert and Ian Clifford. 1993. Achieving Accurate Results With a Circuit Simulator. In *IEE Colloquium on SPICE: Surviving Problems in Circuit Evaluation*.
[29] Tung Thanh Le Le Thanh, Rui Ning, Dan Zhao, Hongyi Wu Wu, and Magdy Bayoumi. 2017. Optimizing the heterogeneous network on-chip design in manycore architectures. In *Proceedings of the 30th IEEE System-on-Chip Conference (SOCC'17)*.
[30] Mieszko Lis, Keun Sup Shim, Myong Hyon Cho, Pengju Ren, Omer Khan, and Srinivas Devadas. 2010. DARSIM: A parallel cycle-level NoC simulator. In *Proceedings of the 6th Annual Workshop on Modeling, Benchmarking and Simulation (MoBS'10)*.
[31] Asit K. Mishra, N. Vijaykrishnan, and Chita R. Das. 2011. A case for heterogeneous on-chip interconnects for CMPs. *SIGARCH Comput. Arch. News* 39, 3 (2011), 389–400.
[32] Thomas Moscibroda and Onur Mutlu. 2009. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th Annual International Symposium on Computer Architecture (ISCA'09)*.
[33] Laurence W. Nagel. 1975. *SPICE2: A Computer Program to Simulate Semiconductor Circuits*. Ph.D. Dissertation. EECS Department, University of California, Berkeley.
[34] Steven Nowick and Montek Singh. 2011. High-performance asynchronous pipelines: An overview. *IEEE Des. Test Comput.* 28, 5 (2011), 8–22.
[35] Freescale Semiconductor. 2014. *MPC8548E PowerQUICC III Integrated Processor Hardware Specifications*. Technical Report. Freescale Semiconductor.
[36] Andi Skende. 2016. Introducing "parker": Next-generation tegra system-on-chip. In *Proceedings of the IEEE Hot Chips Symposium (HCS'16)*.
[37] Jens Sparso and Steve Furber. 2010. *Principles of Asynchronous Circuit Design: A Systems Perspective* (1st ed.). Springer.
[38] Y. Thonnart, X. Tran, P. Vivet, E. Beigne, F. Clermidy, and J. Durupt. 2009. An asynchronous low-power innovative network-On-chip including design-for-test capabilities. In *Proceedings of the International Conference on Advanced Technologies for Communications (ATC'09)*.
[39] Yvain Thonnart, Pascal Vivet, and Fabien Clermidy. 2010. A fully-asynchronous low-power framework for GALS NoC integration. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE'10)*.
[40] E. Tsirogiannis, G. Theodoropoulos, D. Chen, Q. Zhang, L. Janin, and D. Edwards. 2006. *A Framework for Distributed Simulation of Asynchronous Handshake Circuits*. Vol. 2006. IEEE Computer Society, United States, 214–222.
[41] Rafael Ubal, Julio Sahuquillo, Salvador Petit, and Pedro Lopez. 2007. Multi2Sim A simulation framework to evaluate multicore-multithreaded processors. In *Proceedings of the Symposium on Computer Architecture and High Performance Computing (SBAC-PAD'07)*. IEEE Computer Society.
[42] András Varga and Rudolf Hornig. 2008. An overview of the OMNeT++ simulation environment. In *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools'08)*.
[43] Sneha N. Ved, Arjun Gour, Aparna Arya, and Joycee Mekie. 2016. A holistic comparison of static VC allocation versus dynamic VC allocation based NoC routers. In *International Conference on Emerging Electronics (ICEE'16)*.

[44] Hangsheng Wang, Xinping Zhu, Li-Shiuan Peh, and Sharad Malik. 2002. Orion: A power-performance simulator for interconnection networks. In *Proceedings of the 35th Annual International Symposium on Microarchitecture, Istanbul, Turkey, November 18-22, 2002 (MICRO'02)*. 294–305.
[45] Stephen Williams. 2013. Icarus Verilog Status and Goals.
[46] Young Jin Yoon, Paolo Mantovani, and Luca P. Carloni. 2017. System-level design of networks-on-chip for heterogeneous systems-on-chip. In *Proceedings of the 11th IEEE/ACM International Symposium on Networks-on-Chip (NOCS'17)*.

Received December 2017; revised July 2018; accepted July 2018