An analysis and Simulation Tool of Real-Time Communications in On-Chip Networks: A Comparative Study

Chawki Benchehida, Mohammed Kamel Benhaoua, Houssam Eddine Zahaf, Giuseppe Lipari

An analysis and Simulation Tool of Real-Time Communications in On-Chip Networks

A Comparative Study

Chawki Benchehida
benchehida.chawki@edu.univ-oran1.dz
Univ. Oran1, LAPECI Lab
Oran, Algeria

Mohammed Kamel Benhaoua
k.benhaoua@univ-mascara.dz
Univ. Mustapha Stambouli, Mascara
Univ. Oran1 , LAPECI Lab
Oran, Algeria

Houssam-Eddine Zahaf
houssam-eddine.zahaf@univ-lille.fr
Univ. Lille - CRIStAL
Centrale Lille, UMR 9189
Lille, France

Giuseppe Lipari
giuseppe.lipari@univ-lille.fr
Univ. Lille - CRIStAL
Centrale Lille, UMR 9189
Lille, France

Abstract

This paper presents Real-Time Network-on-chip-based architecture Analysis and Simulation tool (ReTiNAS), with a special focus on real-time communications. It allows fast and precise exploration of real-time design choices onto NoC architectures.

ReTiNAS is an event-based simulator written in *Python*. It implements different real-time communication protocols and tracks the communications within the NoC at cycle level. Its modularity allows activating and deactivating different NoC components and easily extending the implemented protocols for more customized simulations and analysis.

Further, we use ReTiNAS to perform a comparative study of analysis and simulation for different communication protocols using a wide set of synthetic experiments.

1 Introduction

The evolution and development of semiconductor technology has made possible the integration of billions of transistors on a single chip. With this technological explosion, designers are able to develop Integrating Complex (ICs) functional elements into a single chip, known as a Multi-Processor System-on-Chip (MPSoC).

MPSoCs contain multiple processing elements (PEs) and are typically classified into homogeneous and heterogeneous. A homogeneous MPSoC contains identical PEs, whereas different types of PEs are integrated in an heterogeneous MPSoC. It provides increased parallelism towards achieving high performance. In such a system, support for on-chip communication is required. The first-generation MPSoCs used buses, hierarchical buses, a point-to-point approach to guarantee the exchange of information between the different components. With the increasing complexity of current and future applications, the number PEs has been increased whereas the main memory is still to be unique. As bus access is exclusive, buses are often subject of high contention, this limits the scalability and becomes quickly a bottleneck of high performances. On-chip packet-switched networks have been proposed as an alternative solution for complex networks. The Network-on-Chip (NoC) communication has been introduced to be power efficient and scalable interconnection to support communication among the PEs.

Real-time systems are usually found in highly critical systems such as avionics, aeronautics, etc. These systems must react to the evolution of the environment. Typically, they have to capture data via several sensors (Cameras, pressure, temperature, etc), process them and finally react to environment state via actuators. To ensure safety, these steps, called also tasks, have to finish within a given time window. In a typical real-time system, several tasks compete for different resources and may also share data.

When executing real-time tasks on a NoC-based architecture, the shared data has to be routed between PEs where communicating tasks are allocated. The needed time to route data from its source to its destination is called communication latency. Latency has to be bounded to ensure that each task instance has been executed without violating the real-time constraints (within its time window).

NoC components (routers and network interfaces, etc) are designed to maximize network utilization without taking into account predictability and temporal behavior of communications, which make them not suitable to real-time systems. An arbitration schema is required to control the access of communication links between PEs, where this mechanism increases the complexity of the NoC. Several works in real-time community have proposed architectural modifications

Ewili'19, October 17, 2019, NY, USA

to reduce the worst case of latency bounds. However, these works are not properly compared against each other, due to a lack of tools (especially simulation).

Contributions This paper presents an event-based simulator and analysis tool for periodic and sporadic real-time communications, allowing to compare the different approaches proposed in the literature against each other by simulation and analysis. We provide also a comparative study of fixed priority and time division arbitration protocols and their impact on latency and schedulability.

The remainder of this paper is organized as follows: in the next section, we report NoC communication mechanisms. Section 3 is reserved to present architecture and communications models. Our Simulator is briefly described in Section 4. Section 5 presents the different approaches to analyze the behavior of fixed priority and TDMA arbitration protocols provided by our tool. Results are discussed in Section 6, we draw conclusion in Section 7.

2 NoC switching and routing mechanisms

Each communication consists of a message, communication source and destination. First, each message $\mathcal{M}_i$ is decomposed into a set of packets ($\mathcal{M}_i = \{\mathsf{P}_{i1}, \mathsf{P}_{i2}, \cdots\}$), further, packets are forwarded separately from a router to another.

Wormhole switching is the mechanism that describes how a packet moves forward from a router to another. In the wormhole switching, each packet P is broken into small pieces called FLITs[1], $\mathsf{P} = \{\mathsf{F}_1^P, \mathsf{F}_2^P, \cdots \mathsf{F}_n^P\}$.

The first flit F_1^P, called the header flit, holds needed information to packet routing (for example, the destination address) and sets up the behavior of all other flits associated within the same packet. Final flit, F_n^P is called the tail flit. Between the header and the tail flit, flits are called body flits.

In wormhole switching, flits are stored in VCs[2]. Each VC is either idle or allocated to only one packet. A header flit can be forwarded to the next router if at least next router has one idle VC. The VC allocator decides where each packet is stored (selects the idle VC for the header flit). When the VC is selected, the header flit locks the VC. Body and tail flits can be forwarded to the same VC as the header, using a credit-based flow control. When the tail flit is routed, it frees the latest VC it has occupied.

In a NoC architecture where each router is composed by one VC per port, if two header flits or more are blocked in a circular dependency, it may lead to a deadlock. Thus, using multiple virtual channels allows to reduce wormhole blocking.

Routing is an operation performed in router to determine which is the next hop of packets. In this paper, we focus only on XY routing. The packets are first transferred in X-direction and then in Y-direction in order to transfer them from the source router to the destination router.

[1]FLow control unITs
[2]Virtual Channels

3 System model

Network on Chip are tightly coupled with computing elements such processors, accelerators, etc. When executing real-time applications on NoC-based architecture, tasks are allocated onto cores such that all real-time requirements are respected. The respect of real-time constraints implies achieving real-time communications in a bounded time. In this paper, we are not interested in task allocation, we focus only on real-time communications. In this section, we present hardware architecture design and task models used in the rest of this paper.

3.1 Architecture model

3.1.1 NoC topology and architecture

We model a NoC architecture $\mathcal{A}$ as a set of $m \times m$ routers. Routers are connected to each other in a 2D-mesh topology (see Figure 1). Each Router is connected to its left, right, top, bottom neighbor except those on the edges.

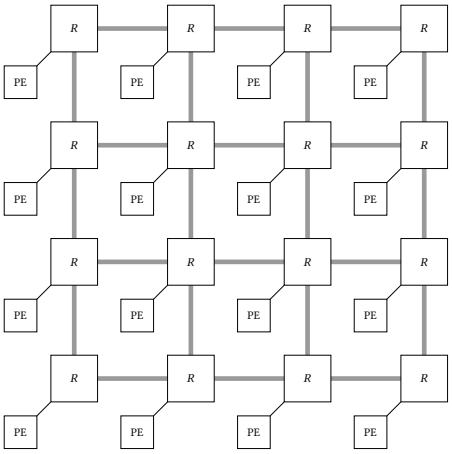


Figure 1. 2D-mesh NoC architecture

$\mathcal{R}_{jm}$ denotes the router at row j and column m. For example $\mathcal{R}_{22}$ denotes the router in the second line and second column. It has for neighbor $\mathcal{R}_{21}$ on the left, $\mathcal{R}_{23}$ on the right, $\mathcal{R}_{12}$ on the top, $\mathcal{R}_{32}$ on the bottom. Routers are linked between them by an uni-directionals links $\lambda_{(i,j)(m,n)}$, where this latter is communicating link from $\mathcal{R}_{ij}$ to $\mathcal{R}_{mn}$.

3.1.2 Router architecture

A router is the main unit in a network-on-chip. Mainly it has k ports, one for each neighbor. In 2D-mesh, each router has 5 in-ports and 5 out-ports connected to its neighbors. The fifth port is local port and connected to the local PE. Figure 2 presents a typical router architecture. Each router is composed of:

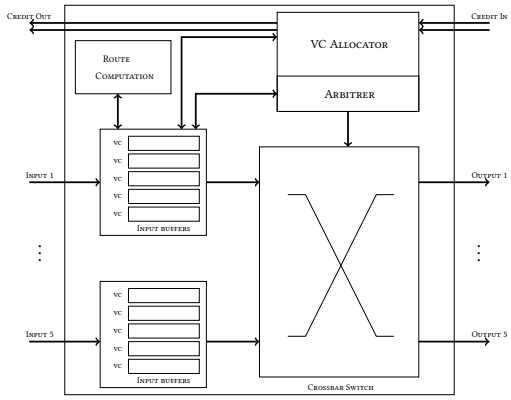


Figure 2. 2D-mesh NoC architecture

- In/Out-ports: are the physical media that links a router with its neighbor routers.
- Virtual Channels (VC): are message buffers. It can contain a fixed number of flit arriving from a neighbor, stored for a while, before being sent to its next destinations (routed). the number of VC per port denoted by |VC| allows a router to support multiple communications using the same port at the same time
- VC Allocator: is the entity responsible of selecting for a given packet, the VC where it is going to be stored.
- Route Computation: is the unit responsible to select the output port for any given packet. Here is implemented XY routing.
- Crossbar: is the unit able to route the non-conflicting communications. By conflicting communication, we denote the packets available at the same time in a given router and need to be routed using the same output port
- Arbiter: the unit that *schedules* outports for conflicting communications. It can be configured to by selecting an arbitration policy to ensure tighter bounds of latency for real-time communications.

3.2 Communication model

Real-time tasks are recurrent. Liu and Layland [9] are the first to model recurrence in real-time systems by defining a real-time task by its deadline, period and offset. The Liu and Layland model is the most used in real-time community and industry. We use similar model for real-time communications. Let $\mathcal{T}$ denote a set of n communications $\mathcal{T} = \{C_1, C_2, \cdots, C_n\}$. Each communication is sporadic and can generate an infinite number of recurrent messages. It is characterized by $C_i = (\mathcal{M}_i, \mathsf{D}_i, \mathsf{T}_i, \mathcal{R}_s, \mathcal{R}_d)$ where:

- $\mathcal{R}_s, \mathcal{R}_d$ are the communication source, and destination routers
- $\mathcal{M}_i$ is the message size sent from $\mathcal{R}_s$ to $\mathcal{R}_d$.
- T_i is the communication period. It represents the minimum arrival time between two communications. Thus, the communication $j+1$ can not start before at least T_i time from the arrival of communication j.
- D_i is the communication relative deadline. The j^{th} communication from $\mathcal{R}_s$ to $\mathcal{R}_d$ has to be finished within the time interval $[\mathsf{a}_{i,j}, \mathsf{a}_{i,j} + \mathsf{D}_i]$ where $\mathsf{a}_{i,j}$ is the time where communication C_i is requested from the router.

In our tool, task parameters are specified using YAML input file.

4 Real-time Communication simulator

Simulation tools allow faster exploration of design space and quick evaluation of the design choices performance. Recently, a lot of simulators [1–5, 7, 8, 10, 11, 13] have been proposed to explore design choices in NoC-based architectures at different abstraction and precision levels. For example, GPNoCSIM [7] and DynaMapNocSim [2] are event-based simulators written in JAVA, the first focuses on communications, whereas the second focuses more on the task allocation, both at high level of abstraction. Hermes [11], is low-level simulation tool withen in VHDL. It allows to emulate design choices on FPGA boards, however it is time consuming to explore design choices and evaluate their performances.

However, none of the simulators, cited above, offers a support for real-time communications, neither periodicity or recurrence in general. The latter are designed for non-critical systems and need lot of modifications to make them support real-time communication protocols. Thus, we propose a new simulation tool for real-time communication protocols.

Our simulator is modular, and extensible. A first version is available[3] and is still under continual upgrading and development to include extra-features. In this section, we describe how the simulator has been designed.

4.1 Packages

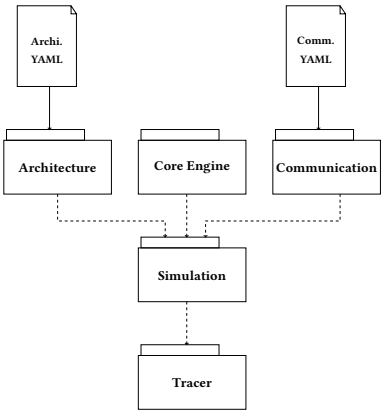


Figure 3. Package diagram

Our simulation tool is compound of 5 packages, detailed in the follow :

[3] https://github.com/chawki27000/retina-sim

Architecture : It contains all the classes and structures to define a NoC. We focus mainly on 2D Mesh topology. However, our design can be extended to specify other topologies like torus and ring.

Communication: This package defines the router communication structures and their parameters. Allowing to define periodic and aperiodic communications, message decomposition, structure and serveral extra-real-time parameters.

Core engine: Here are implemented all the algorithms that contributes in NoC functioning. They are mostly implementated by different interfaces (CROSS BAR, VC allocator, Arbitrer, $\cdots$).

Simulation : The simulation package contains the simulation core. It is responsible for events and time management. It contains discrete event-based simulation engine. The simulation engine can be re-used for any other simulation purposes.

Tracer : Responsible of registering at cycle level, all actions taken onto each router and the state of each VC at each time instance are save in a log file. It allows also to automatically generate formatted results and some predefined plots using PGF-plots.

4.2 NoC & Simulation Engines

Our simulator handles three types of events :

- MESSAGE_ARRIVAL : This event occurs to signal a communication between two routers. It starts from message splitting to reach flit granularity until generate next events.
- SEND_HEAD_FLIT : This event handles flit header forwarding, by defining the next hop router, reserving an idle VC, triggering arbitration (if conflict occurs) and generate the next events if all is done without errors.
- SEND_BODY_TAIL_FLIT : Finally, this event handles body or tail flit. It checks free space in the allocated VC, blocking flit sending if no space available and releasing VC if the current flit is tail.

Algorithm 1 shows different steps and main function calls of the simulator. It starts by parsing a NoC settings and communication scenario by instanciating all periodic or aperiodic communications. Further, it sorts all events and loops on them one by one. The clock is updated when the event is handled. The simulation ends when simulation time reaches the hyper-period or events list is empty.

Algorithm 1 Simulation

```
1:  noc_f: YAML FILE
2:  task_f: YAML FILE
3:  parse_noc(noc_f)
4:  create_events(task_f)
5:  sort_events_time(task_f)
6:  while (event_list ≠ ∅) do
7:      e = select_next_event()
8:      update_clock()
9:      switch e do
10:         case MESSAGE_ARRIVAL :
11:             Process_message(e)
12:         case SEND_HEAD_FLIT :
13:             send_header_flit(e)
14:         case SEND_BODY_TAIL_FLIT:
15:             send_body_tail_flit(e)
16:     if (sim_time_finished) then
17:         empty_event_list()
18:     end if
19: end while
```

5 Analysis

In this section, we present priority-based and TDMA-based arbitration mechanisms and their analysis.

5.1 Fixed priority

Shi et al. [12] propose to assign a priority to virtual channels. Therefore a communication in VC of priority p, is selected by the arbiter before any communication in all VCs of priority less than p. Moreover, if the communication in VC of priority p has already started, it can be interrupted by communications in VCs of higher priority, allowing preemptions. Once the high priority communication finishes, the low priority communication resume their forwarding in a classical preemption scheme. Authors in [12] provided worst case communication latency bounds analysis. Tighter bounds has been provided by Xiong et al. [14] and [15] by distinguishing two types of interference : *Upstream* and *Downstream*. In a 2D-NoC topology, Upstream interference is caused by conflicting messages arriving from the south port, whereas the downstream interference is caused by incoming communication from north port. Equation 1 have been proposed by the authors of [15] to compute a communication latency bounds.

$$R_i = \sum_{\forall \tau_j \in S_i^D} \lceil \frac{R_i + J_j + I_{ji}^U}{T_j} \rceil (C_j + I_{ji}^D) + C_i \qquad (1)$$

Equation 1 is iterative:

It starts by assuming $R_i^{(0)} = \sum_{\forall \tau_j \in S_i^D} (C_j + I_{ji}^D) + C_i$, where

- S_i^D is a set of messages that constitutes a direct interference
- I_{ji}^U I_{ji}^D is the set of conflicting messages belonging to Upstream and Downstream (indirect interference)

This equation converges if fixed point is found ($R_i^{(n+1)} = R_i^{(n)}$) or if latency is already greater than the deadline, resulting to a deadline miss ($R_i^{(n)} > D_i$).

Although this approach is *easy* to implement and analyze, it presents major limitations. First, preempting a communication can be a costly operation. In fact, the router is forced to create and schedule a tail FLIT for the preempted message so it can continue onward its routing and a new head FLIT for the FLITs that are still not yet forwarded. Moreover, a real implementation of such solutions requires as much VCs per input port as number of priorities (tasks). However, increasing the number of VC (which are mainly buffers) increases drastically the chip size and lead to heat dissipation and voltage problems. One solution may be to limit the number of scheduled tasks or to manage the priorities in a hierarchical schedule scheme.

5.2 Time division multiple access

Abbreviated by TDMA, the second main approach aims to share output port between conflicting communications based on time sharing. Therefore, each VC has its own service time slots, where FLITs within that VC are forwarded. Several works have been interested in optimizing the time slot size and slot assignment. A exhaustive survey can be found in [6].

Under TDMA, each communication is achieved in isolation to the others. Its latency can be computed as shown in Equation 2.

$$R_i = \frac{L_i}{n_{slot}} \cdot \frac{\Delta}{\delta_i} + H_i \tag{2}$$

Where :

- L_i : number of flits in the message.
- n_{slot} : The amount of data sent in one slot (1 Flit by default).
- Δ / δ_i : The total number of slots in a *TDMA cycle* / the assigned slot number.
- H_i : Hop number between $\mathcal{R}_s$ and $\mathcal{R}_d$.

This approach is more complex and requires implementing timers and their synchronization mechanisms in the routers. However, it provides isolation of FLIT forwarding, therefore prevents "miss-behaving" communications from monopolizing the network. Furthermore, it does not require other modifications to VC structures, nor to arbitration protocols. However, communication-to-VC assignment mechanisms must be achieved offline.

6 Experiments

In this section, we present a wide set of synthetic simulations to study performances of fixed-priority-based approach against TDMA based approaches in terms of worst case latency bounds and resource augmentation. ReTiNAS is used to simulate the real-time task communication behavior. Communications latency change drastically when all conflicting communications are active at the same time. Therefore, we start by describing how conflicting communications are generated.

6.1 Conflicting communications generation

First, a communication com is selected between src and dst . Further, the route between src and dst is computed using XY-routing algorithm. Later, stressing communications which has a goal to create contention in either X-axe or Y-axe or both, are iteratively generated until reaching an input *contention rate* threshold.

The contention rate is computed as follows:

$$\sum_{\tau_i \in \text{ConflictSet}} \frac{M_i}{T_i} \tag{3}$$

where ConflictSet represents the higher priority tasks that share at least a link with the current task as depicted in Figure 4.

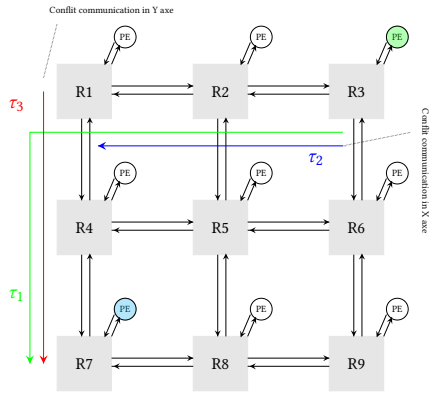


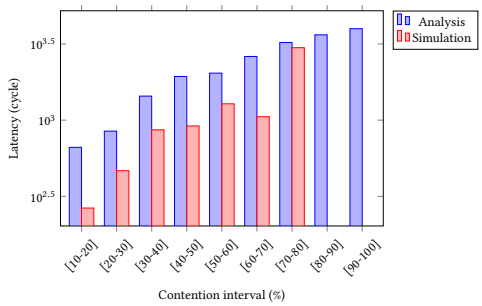
Figure 4. Interference on the message path

6.2 Simulation

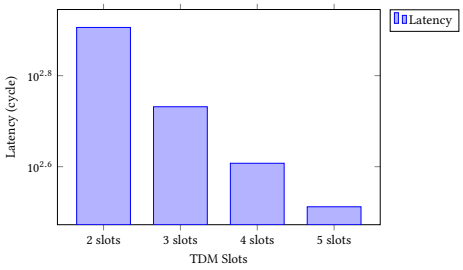
We perform a wide set of experiments using ReTiNAS, using a personal computer including Intel i5-7200U CPU and 8GB of RAM. All experiments where achieved on the NoC configuration summarized on Table 1.

Topology	4x4 2D-Mesh
VC per InputPort	6
Buffer size per VC	10 Flits
Periods (cycle)	[1000, 1500, 2000, 3000, 4000, 6000]
TDM Slots	[4, 2, 3, 5, 3, 3]
Message	8 Packets (10 Flits each)
Number of hop	5

Table 1. NoC configuration and Communication detail



(a) Simulation/Analysis in Preemptive Priority Arbitration



(b) Analysis by TDMA Arbitration

Figure 5. Experimentation results in diffrent arbitration mode

Figure 5a reports the worst case latency obtained by simulation against the one obtained by analysis as a function of contention rate for fixed priority approach. When the contention is low, the analysis tend to compute very large latency bounds compared to the measured latency using the simulator. In fact, the analysis assume always the worst case of task arrivals, therefore a congestion level that may never be reached when tasks execute. The more contention, the high latency is, in both simulation and analysis. In fact, in such scenario, the worst scenarios can often happen when all tasks are activated, therefore the simulation worst bounds are close to the analytical ones. However, analysis is still slightly over-estimating the worst case latency bounds.

TDMA is a contention-free arbitration approach as all communications are executed in isolation. Therefore, it is not interesting to study the impact of congestion on latency itself. Thus, in Figure 5b, we report the latency using *TDMA* approach as a function of the time slot size. As expected, the bigger time slot size, the shorter latency is. However, communications may not be able to be served using unlimited time-slot size. The problem is to define the *exact* time slot per time period for a given task to respect its real-time constraints.

Figure 6 represents the average VC number required, for fixed priority and TDMA, to respect deadlines for the same tasks as a function of contention rate. This allows us to compare the efficiency of each approach regarding the respect of real-time constraints. We highlight that we modify the time-slot size to have schedulable tasks for TDMA. Therefore,

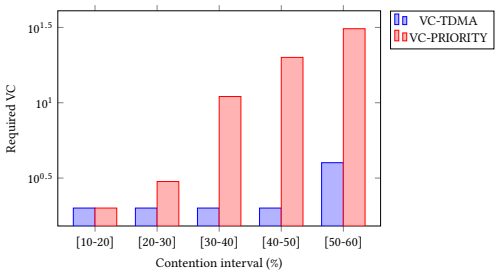


Figure 6. NoC Resource Augmentation

even if results reported here show the efficiency TDMA and fixed priority, it does not allow a fair comparison of TDMA against fixed priority.

When contention is low, we can see that TDMA approach need the same number of VCs as fixed priority. When the contention rate is increased, the congestion increases, therefore more of higher priority messages are scheduled and more VCs are needed to keep the latency less than the deadline. TDMA is not contention sensitive, therefore always requires less VCs compared to fixed priority. The gap between both keeps increasing as the contention is increased.

7 Conclusion

In this paper, we presented the design and implementation of a real-time network-on-chip communication simulator and analysis tool. We provided also an overview of techniques to perform real-time communication in a NoC architectures. We presented a comparative study of TDMA and fixed priority approachs as a function of worst case latency and resource augmentation bounds. As future work, we would like to investigate exact solutions for budgeting VCs including the task allocation problem for TDMA.

Acknowledgments

This research was supported in part by MESRS, Algeria and by PHC Tassili project 19MDU213 and by the PRIMA WATERMED 4.0 project. We would like also to aknowledge the support of Benyamina Abou Elhassen, a full professor of Oran university, for his administrative support.

References

[1] Yaniv Ben-Itzhak, Eitan Zahavi, Israel Cidon, and Avinoam Kolodny. 2012. HNOCS: modular open-source simulator for heterogeneous NoCs. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*. IEEE, 51–57.

[2] Mohammed Kamel Benhaoua, AmitKumar Singh, Abou El Hassan Benyamina, and Pierre Boulet. 2015. DynMapNoCSIM: A Dynamic Mapping SIMULATOR for Network on Chip based MPSoC. *Journal of Digital Information Management* 13, 1 (2015).

[3] 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.

[4] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. 2004. QNoC: QoS architecture and design process for network on chip. *Journal of systems architecture* 50, 2-3 (2004), 105–128.

[5] Fabrizio Fazzino, Maurizio Palesi, and David Patti. 2008. Noxim: Network-on-chip simulator. *URL: http://sourceforge. net/projects/noxim* (2008).

[6] Salma Hesham, Jens Rettkowski, Diana Goehringer, and Mohamed A. Abd El Ghany. 2017. Survey on Real-Time Networks-on-Chip. *IEEE Transactions on Parallel and Distributed Systems* 28, 5 (May 2017), 1500–1517. https://doi.org/10.1109/TPDS.2016.2623619

[7] H. Hossain, M. Ahmed, A. Al-Nayeem, T. Z. Islam, and M. M. Akbar. 2007. Gpnocsim - A General Purpose Simulator for Network-On-Chip. In *2007 International Conference on Information and Communication Technology*. 254–257. https://doi.org/10.1109/ICICT.2007.375388

[8] Mieszko Lis, Keun Sup Shim, Myong Hyon Cho, Pengju Ren, Omer Khan, and Srinivas Devadas. 2010. DARSIM: a parallel cycle-level NoC simulator. In *MoBS 2010-Sixth Annual Workshop on Modeling, Benchmarking and Simulation*.

[9] C. L. Liu and James W. Layland. 1973. Scheduling Algorithms for Multiprogramming in a Hard-Real-Time Environment. *J. ACM* 20, 1 (Jan. 1973), 46–61. https://doi.org/10.1145/321738.321743

[10] Leandro Möller, Leandro Soares Indrusiak, and Manfred Glesner. 2009. NoCScope: A graphical interface to improve Networks-on-Chip monitoring and design space exploration. In *Design and Test Workshop (IDT), 2009 4th International*. IEEE, 1–6.

[11] Fernando Moraes, Ney Calazans, Aline Mello, Leandro Möller, and Luciano Ost. 2004. HERMES: an infrastructure for low area overhead packet-switching networks on chip. *INTEGRATION, the VLSI journal* 38, 1 (2004), 69–93.

[12] Z. Shi and A. Burns. 2008. Real-Time Communication Analysis for On-Chip Networks with Wormhole Switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*. 161–170. https://doi.org/10.1109/NOCS.2008.4492735

[13] Kartika Vyas, Naveen Choudhary, and Dharm Singh. 2013. NC-G-SIM: A Parameterized Generic Simulator for 2D-Mesh, 3D-Mesh & Irregular On-chip Networks with Table-based Routing. *Global Journal of Computer Science and Technology* (2013).

[14] Qin Xiong, Zhonghai Lu, Fei Wu, and Changsheng Xie. 2016. Real-Time Analysis for Wormhole NoC: Revisited and Revised. In *Proceedings of the 26th edition on Great Lakes Symposium on VLSI - GLSVLSI '16*. ACM Press, Boston, Massachusetts, USA, 75–80. https://doi.org/10.1145/2902961.2903023

[15] Qin Xiong, Fei Wu, Zhonghai Lu, and Changsheng Xie. 2017. Extending Real-Time Analysis for Wormhole NoCs. *IEEE Trans. Comput.* 66, 9 (Sept. 2017), 1532–1546. https://doi.org/10.1109/TC.2017.2686391