Real-time Communications in On-Chip Networks

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

Real-time Communications in On-Chip Networks

Chawki Benchehida & Mohammed Kamel Benhaoua
Univ. d'Oran 1 - LAPECI Laboratory
B.P 1524 ,El M'Naouer
Oran, Algeria
benchehida.chawki@edu.univ-oran1.dz
k.benhaoua@univ-mascara.dz

Houssam-Eddine Zahaf & Giuseppe Lipari
Univ. Lille - CRIStAL
Centrale Lille, UMR 9189
Lille, France
{houssam-eddine.zahaf, giuseppe.lipari}@univ-lille.fr

ABSTRACT

Networks on-Chip (commonly known as NoC) have emerged as a promising solution to bus congestion and bus unpredictable share in multi/many processor architectures. Real-time systems are highly critical and need to be as predictable as possible to ensure safety. This paper presents an analysis/simulation tool design for the execution of periodic and sporadic real-time tasks on NoC-based architectures, with a special focus on real-time communications. We propose an event-based simulator written in *JAVA*. The simulator implements different real-time communication protocols and tracks the different communications at cycle level. The simulator modularity allows activating and deactivating different NoC components and easily extending the implemented protocols for more customized simulations and analysis.

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).

An MPSoC contains multiple processing elements (PEs) and can typically be classified into homogeneous and heterogeneous. The first contains identical PEs whereas different types of PEs are integrated in the second. The first-generation MPSoCs used buses to allow information exchange between components. With the increase of PEs in a single chip, the bus is highly contented, which limits the scalability and becomes quickly a bottleneck for high performances. Networks on-Chip (NoC) has been proposed as an alternative solution for scalable interconnection between PEs and power efficiency.

On the other hand, real-time systems are usually found in highly critical systems such as avionics, aeronautics, $\cdots$. These systems should adapt their behavior onto the evolution of the environment. Typically, they have to capture data via several sensors (Cameras, pressure, temperature, ...), process them and finally react to environment state via actuators. To ensure safety, these steps, called also tasks, have to finish their execution within a given time window. In a typical real-time system, several tasks are in concurrence on 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, $\cdots$) 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. Routing real-time communications onto classical NoC, will lead to a loss of the urgency (priority) of tasks and may lead to large latencies for highly critical tasks. Several works in real-time community have proposed architectural modifications to reduce worst case of latency bounds. However these works are not properly compared against each other from practical perspective, due to a lack of tools (especially simulation). Currently NoCs research community has developed several simulators. These simulation tools are not fundamentally designed for real-time systems. Therefore they do not offer support and mechanisms for real-time communications.

This paper presents an event-based simulation and analysis tools for periodic and sporadic real-time communications. The simulator design is detailed, with a special focus on NoC functionalities and how they can be extended. We also provide a brief comparison between different NoC communication strategies proposed in the literature of real-time systems.

2. OVERVIEW OF ROUTING IN NOC

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. Each VCs is either idle or allocated to only one packet. A header flit

[1]FLow control unITs

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 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. A packet is routed onto the next routers on the X axes until reaching the destination Y axes and then it starts being forward on the Y axes until reaching destination X axes. XY is a deterministic and predictable routing algorithm.

3. SYSTEM MODEL

Network on Chips are tightly coupled with computing elements such processors, accelerators, etc. When executing real-time applications on NoC-based architectures, 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 communication. 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.

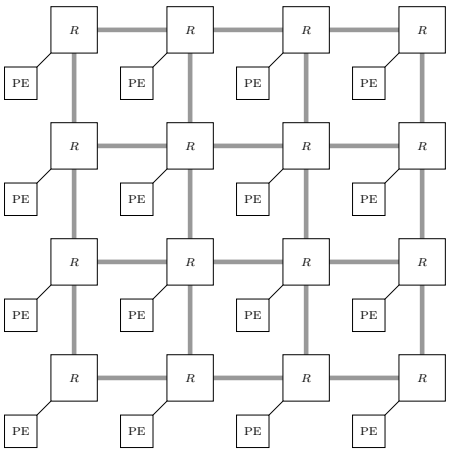

Figure 1: 2D-mesh NoC architecture

$\mathcal{R}_{jm}$ 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 left, $\mathcal{R}_{12}$ on the top, $\mathcal{R}_{32}$ on the bottom.

3.1.2 Router architecture

A router is the main unit in a network-on-chip. Mainly it has of k ports, each for a 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 compound of:

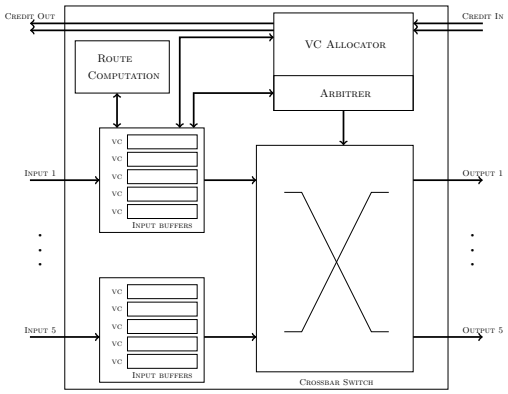

Figure 2: 2D-mesh NoC architecture

- In/Out-ports: are the physical media that links a router with its neighbor routers.
- Virtual Channels (VC): are message buffers. It contains a flits arriving from a neighbor, stored for a while, before being sent to its next destinations (routed). the number of VC per port denoted by $|\mathsf{VC}|$ 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. This is one of the main units that will be modified later to ensure tighter bounds of latency for real-time communications.

3.2 Communication model

Real-time tasks are recurrent. Liu and Layland 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 communication analysis. Let $\mathcal{T}$ denote a set of n communication $\mathcal{T} = \{\mathcal{C}_1, \mathcal{C}_2, \cdots, \mathcal{C}_n\}$. Each communication is sporadic and can generate an infinite number of message exchanging from source router to destination router and is characterized by $\mathcal{C}_i = (\mathcal{M}_i, \mathsf{D}_i, \mathsf{T}_i, \mathcal{R}_s, \mathcal{R}_d)$ where:

- $\mathcal{R}_s, \mathcal{R}_d$ are the communication source, and destination routers
- $\mathcal{M}_i$ is the maximum size of communication between $\mathcal{R}_s$ and $\mathcal{R}_d$ for communication i.
- T_i is the communication period. It represents the minimum arrival time between two communication. 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 $\mathcal{C}_i$ is requested from the router.

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

Real-time communications need to be analyzed to compute the worst case communication latency and ensure that each communication finishes within its deadline. Task and communication allocation are built according to the worst case latency obtained by analysis(WCLbA). If WCLbA is hugely greater than average case latency, the platform utilization is drastically degraded. In this paper, we call the distance between worst case and average case latency-pessimism. To compare real-time protocols for NoC architectures, it is sufficient to compare the WCLbA they provide. However, this comparison is not sufficient since it does not allow you to have an idea about resource occupation. Here, we need to refer to simulation.

In the real-time community, several works have proposed architectural modifications to reduce latency-pessimism, here we present some of them and how they are implemented in our simulator. For space constraints, we overview the main ideas of each.

Giuseppe et al. in [7] compute the worst case latency without any additional support to router structures. The authors propose a formulation of an end-to-end communication latency analysis including latency of network access (from processor to router) and network latency. The analysis presented in [7] does not need any further modifications into our simulation and router structure. However theoretically it provides larger latency-pessimism.

Burns et al. in [11] propose to assign priorities to virtual channels. For a given set of n communications, we have n virtual channel per each port. The VC allocators assign each communication into its corresponding VC. This solution brings some realization problems : the increase of surface and cost of chip production due to large VC number, communication preemption overheads and context switch costs by header and tail flits recreation.

Bandwidth reservation techniques comes as solution in the middle. They don't impose a lot of critical structural modifications in the router and can grant a service time for each communication.

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 [8, 12, 3, 4, 9, 5, 2, 1, 10, 6] have been proposed to explore design choices in NoC-based architectures at different abstraction and precision levels. For example, GPNoCSIM [6] 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 [10], 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 these 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[2] 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

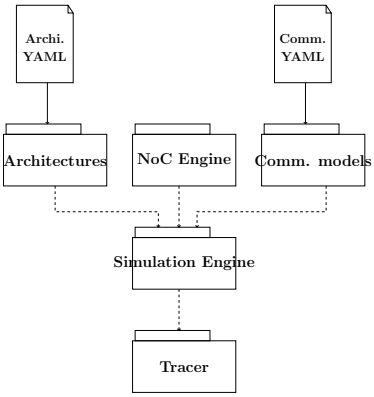

Figure 3: Package diagram

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

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

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

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

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 handle 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 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
```

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 is reached or event list is empty.

5. CONCLUSION

In this paper, we presented a design of real-time simulator and we provide briefly an overview of techniques to perform real-time communication in a NoC architectures. We present some research threats and how our simulator can help in achieving architectural modifications without a lot of programming efforts. Due to space constraints, we did not present simulation results. For future work, we are planning to compare the performances of each real-time communications scheduling techniques in terms of latency and energy efficiency. Further, we would like to investigate exact solution for budgeting VCs.

6. REFERENCES

[1] Y. Ben-Itzhak, E. Zahavi, I. Cidon, and A. Kolodny. Hnocs: modular open-source simulator for heterogeneous nocs. In *Embedded Computer Systems (SAMOS), 2012 International Conference on*, pages 51–57. IEEE, 2012.

[2] M. K. Benhaoua, A. Singh, A. E. H. Benyamina, and P. Boulet. Dynmapnocsim: A dynamic mapping simulator for network on chip based mpsoc. *Journal of Digital Information Management*, 13(1), 2015.

[3] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7, 2011.

[4] E. Bolotin, I. Cidon, R. Ginosar, and A. Kolodny. Qnoc: Qos architecture and design process for network on chip. *Journal of systems architecture*, 50(2-3):105–128, 2004.

[5] F. Fazzino, M. Palesi, and D. Patti. Noxim: Network-on-chip simulator. *URL: http://sourceforge.net/projects/noxim*, 2008.

[6] H. Hossain, M. Ahmed, A. Al-Nayeem, T. Z. Islam, and M. M. Akbar. Gpnocsim - a general purpose simulator for network-on-chip. In *2007 International Conference on Information and Communication Technology*, pages 254–257, March 2007.

[7] S. Kumar and G. Lipari. Latency analysis of network-on-chip based many-core processors. In *22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2014, Torino, Italy, February 12-14, 2014*, pages 432–439, 2014.

[8] M. Lis, K. S. Shim, M. H. Cho, P. Ren, O. Khan, and S. Devadas. Darsim: a parallel cycle-level noc simulator. In *MoBS 2010-Sixth Annual Workshop on Modeling, Benchmarking and Simulation*, 2010.

[9] L. Möller, L. S. Indrusiak, and M. Glesner. Nocscope: A graphical interface to improve networks-on-chip monitoring and design space exploration. In *Design and Test Workshop (IDT), 2009 4th International*, pages 1–6. IEEE, 2009.

[10] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. Hermes: an infrastructure for low area overhead packet-switching networks on chip. *INTEGRATION, the VLSI journal*, 38(1):69–93, 2004.

[11] Z. Shi and A. Burns. Real-time communication analysis for on-chip networks with wormhole switching. In *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, pages 161–170, April 2008.

[12] K. Vyas, N. Choudhary, and D. Singh. 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.